

Barème : 1 points par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.

1. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3
- ☐ 4 3 2 1

2. Le type des réels en C est :

- ☐ char
- ☐ real
- ☐ double
- ☐ int

3. Vous utilisez une boucle while quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un for dans le même programme principal

4. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `saisie_utilisateur scanf(%d);`

5. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 16
- ☐ 20
- ☐ 3

6. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ `a = b`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a{n, x} == b{n, x}`
- ☐ `a == b`

7. Si n est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur
- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `scanf("%d", &n);`

8. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
```

```
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\*
- ☐ \*\* \*\*\*
- ☐ \*\* \*\* \*
- ☐ \*\*\*\*

9. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 5
- ☐ 101
- ☐ 4
- ☐ 3

10. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur Majeur

11. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ en parallèle, chacun dans un registre
- ☐ tour à tour, un petit peu à chaque fois
- ☐ tous ensemble
- ☐ chacun son tour, après que le processus précédent a terminé

12. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
- ☐ `for(i=0;i<=5;i=i+1)`
  - ☐ `for(i=1;i<=5;i=i+1)`
  - ☐ `for(i=0;i<5;i=i+1)`
  - ☐ `for(i=1;i<5;i=i+1)`
13. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :
- ☐ `mkdir TP4`
  - ☐ `yppasswd`
  - ☐ `new TP4`
  - ☐ `kwrite TP4`
14. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
- ☐ `char c;`
  - ☐ `char 'c';`
  - ☐ `int char;`
  - ☐ `char "c";`
15. Si x est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :
- ☐ 0.5
- ☐ 1.5
- ☐ 1
- ☐ 0
16. Sur unix (ou linux), la commande `mkdir` permet de :
- ☐ créer un répertoire
  - ☐ créer un fichier texte
  - ☐ ouvrir un fichier texte
  - ☐ changer de répertoire courant
17. Quel est le problème d'un programme comportant les lignes suivantes ?
- ```
while (1)
{
    printf("coucou\n");
}
```
- ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il comporte une boucle infinie
  - ☐ il ne compile pas
  - ☐ il n'affiche rien
18. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ alphabétique
19. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction
- ☐ `loop i;`
  - ☐ `int %d;`
  - ☐ `int k;`
  - ☐ `int loop n;`
20. Le code suivant :
- ```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 0 1 2 3 4
  - ☐ 4 3 2 1
  - ☐ 4 3 2 1 0
  - ☐ 1 2 3 4

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée et définie
- ☐ l'avoir définie
- ☐ l'avoir déclarée
- ☐ avoir défini une constante symbolique de la taille de cette fonction

2. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int n = carre();`
- ☐ `n = carre(int n);`
- ☐ `n = carre(n);`
- ☐ `int carre(2);`

3. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

4. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`
- ☐ `void exposant(double x^n);`

5. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2 3
- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 3 0 1 2
- ☐ 0 1 2 0 1 2

6. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ changer de répertoire courant
- ☐ ouvrir un fichier texte
- ☐ créer un fichier texte
- ☐ créer un répertoire

7. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `int factorielle(int 2);`
- ☐ `printf("%d", factorielle(n));`
- ☐ `n = factorielle();`
- ☐ `n = factorielle(p, q);`

8. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b
- ☐ C
- ☐ B
- ☐ A

9. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

10. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`

11. Le langage C est un langage

- ☐ lu, écrit, parlé
- ☐ compilé
- ☐ interprété
- ☐ composé

12. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse sémantique
- ☐ analyse lexicale
- ☐ analyse syntaxique

13. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char c;`
- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char 'c';`

14. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`
- ☐ `int pgcd(2);`
- ☐ `n = pgcd(n, 3);`
- ☐ `n = pgcd(int p, int q);`

15. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(n<=a) && (n<=b)`
- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(a<n) || (n>b)`

16. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie

17. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(char c);`

18. Un fichier source est :

- ☐ un document illisible pour les humains
- ☐ un document qui doit être protégé
- ☐ un document de référence du système
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur

19. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(double n);`
- ☐ `int factorielle();`

20. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <stdio.h>`
- ☐ `#include <studio.h>`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 0  
☐ 8  
☐ 16  
☐ 4

2. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur  
☐ Arriver à l'heure en cours  
☐ Transférer des données et intructions entre processeur et mémoire  
☐ Écrire des données sur le disque dur

3. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = %d  
☐ j = 0  
☐ j = 5  
☐ j = 4

4. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tour à tour, un petit peu à chaque fois  
☐ chacun son tour, après que le processus précédent a terminé  
☐ en parallèle, chacun dans un registre  
☐ tous ensemble

5. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5  
☐ f(a,b)=8, a=8, b=5  
☐ f(a,b)=8, a=3, b=5  
☐ f(3,5)=8, a=3, b=5

6. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char 'c';  
☐ char c;  
☐ char "c";  
☐ int char;

7. Une variable booléenne est un variable :

- ☐ NaN (not a number, qui n'est pas un nombre)  
☐ à laquelle une valeur vient d'être affectée  
☐ qui est vraie ou fausse  
☐ réelle positive  
☐ jamais nulle

8. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse sémantique  
☐ analyse harmonique  
☐ analyse syntaxique  
☐ analyse lexicale

9. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ int n = carre();  
☐ int carre(2);  
☐ n = carre(int n);  
☐ n = carre(n);

10. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ la variable x vaut  $-\frac{1}{2}$   
☐ le programme affiche \*\*\*\*  
☐ le programme affiche x  
☐ la variable x vaut 16

11. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop  
☐ un point-virgule manquant  
☐ une accolade en trop  
☐ une accolade manquante

12. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A
- ☐ ABCDEF
- ☐ cccccc
- ☐ i

13. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il ne compile pas

14. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 2
- ☐ 8 6 4 2 0
- ☐ 8 6 4 2
- ☐ 0 2 4 6 8

15. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7
- ☐ 111
- ☐ 8
- ☐ 3

16. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a == b
- ☐ a{n, x} == b{n, x}
- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a = b

17. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses cases
- ☐ ses chants
- ☐ ses blocs

18. Vous utilisez une boucle `while` quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal

19. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\* \*\* \* \*\*\*\* \*\*\*\*\*
- ☐ \*\* \*\* \* \* \* \* \* \*
- ☐ \*\*\*\* \*\*\*\*\* \*\*\*\* \*
- ☐ \*\*\*\*\* \*\*\*\*\* \*\*\* \*\*

20. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 3
- ☐ 20
- ☐ 16

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le langage C est un langage

- ☐ lu, écrit, parlé
- ☐ compilé
- ☐ composé
- ☐ interprété

2. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 0 1 1 1
- ☐ 1 2 1 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 0 1 0 1 0 1

3. Vous utilisez une boucle **while** quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un **for** dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1

4. Pour déclarer une fonction **saisie\_utilisateur** qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `saisie_utilisateur scanf(%d);`

5. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction **f** ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

6. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse syntaxique
- ☐ analyse lexicale
- ☐ analyse sémantique
- ☐ analyse harmonique

7. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur
- ☐ Arriver à l'heure en cours

8. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui ne sera pas détectée
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique

9. Si cette erreur apparaît à la compilation :

**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ un caractère interdit en C
- ☐ une directive préprocesseur **#include** manquante
- ☐ une variable non déclarée
- ☐ une faute de frappe dans un appel de fonction

10. Au début de la fonction **main()** on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A
- ☐ C
- ☐ B
- ☐ b

11. L'ordonnancement par tourniquet permet :

- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ de doubler la mémoire disponible
- ☐ d'afficher des ronds colorés à l'écran

12. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 5
- ☐ 4
- ☐ 3

13. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

14. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce `printf`?

- ☐ `j = 0`
- ☐ `j = %d`
- ☐ `j = 4`
- ☐ `j = 5`

15. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ i
- ☐ A
- ☐ cccccc

16. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(double n);`
- ☐ `int factorielle();`
- ☐ `int factorielle(int x);`

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Mineur Majeur
- ☐ rien

18. Le type des réels en C est :

- ☐ `double`
- ☐ `char`
- ☐ `int`
- ☐ `real`

19. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 4
- ☐ 1

20. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tous ensemble
- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ tour à tour, un petit peu à chaque fois
- ☐ en parallèle, chacun dans un registre



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = 0
- ☐ j = %d
- ☐ j = 4
- ☐ j = 5

2. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ (!A || B)
- ☐ (A == TRUE) && (B == TRUE)
- ☐ A && B
- ☐ !(A || B) == (A && !B)

3. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ changer de répertoire courant
- ☐ créer un fichier texte
- ☐ créer un répertoire
- ☐ ouvrir un fichier texte

4. Un bit est :

- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur

5. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses chants
- ☐ ses cases
- ☐ ses champs

6. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 0
- ☐ 8
- ☐ 16
- ☐ 4

7. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu(int char);`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu();`
- ☐ `char afficher_menu(sprintf("menu"));`
- ☐ `double afficher_menu();`

8. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `int saisie_utilisateur();`
- ☐ `saisie_utilisateur(scanf("%d"));`

9. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`

10. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Majeur
- ☐ Mineur

11. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char 'c';`

12. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ retourner un bloc
- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ mettre les blocs en séquence les uns à la suite des autres
- ☐ répéter un bloc tant qu'une condition est vérifiée

13. Le type des réels en C est :

- ☐ `char`
- ☐ `double`
- ☐ `real`
- ☐ `int`

14. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

15. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle();`
- ☐ `n = factorielle(p, q);`
- ☐ `int factorielle(int 2);`

☐ `printf("%d", factorielle(n));`

16. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(struct date_s d);`

17. Un fichier source est :

- ☐ un document qui doit être protégé
- ☐ un document de référence du système
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un document illisible pour les humains

18. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

☐ il risque d'afficher bonjour à la place de coucou

☐ il ne compile pas

☐ il comporte une boucle infinie

☐ il n'affiche rien

19. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x - 1 = racine(x);`

20. Le langage C est un langage

☐ lu, écrit, parlé

☐ interprété

☐ compilé

☐ composé

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses cases
- ☐ ses champs
- ☐ ses chants

2. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ rien
- ☐ Mineur  
Majeur
- ☐ Majeur
- ☐ Mineur

3. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(x * x) - racine(x);`

4. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur  
Majeur

5. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ alphabétique
- ☐ un ordre quelconque

6. L'ordonnancement par tourniquet permet :

- ☐ de doubler la mémoire disponible
- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'afficher des ronds colorés à l'écran

7. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8
- ☐ 3
- ☐ 111
- ☐ 7

8. Pour déclarer une fonction **pgcd** qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int y, int x);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, y);`

9. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `include`
- ☐ `begin`
- ☐ `init`
- ☐ `main`

10. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas

11. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti

12. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction **main()**, alors on peut écrire en début de **main()** :

- ☐ `int toto.n = 3;`

```

☐ int struct toto_s = {3, -1e10};
☐ toto_s struct z = {3, 0.5};
☐ toto_s n, x;
☐ struct toto_s toto;

```

13. Le langage C est un langage

- ☐ composé
- ☐ lu, écrit, parlé
- ☐ compilé
- ☐ interprété

14. Après exécution du programme :

```

1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5

```

- ☐ le bus explose
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 16
- ☐ la case mémoire 8 contiendra 0

15. Soit un programme contenant les lignes suivantes :

```

int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}

```

```

    }
}
printf("j = %d\n", j);

```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 4
- ☐ j = 5
- ☐ j = %d
- ☐ j = 0

16. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme?

- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie

17. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`
- ☐ `void exposant(double x^n);`
- ☐ `double exposant(double x, int n);`

18. Soit le programme principal suivant :

```

int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}

```

appelant la fonction **f** ainsi définie :

```

int f(int a, int b)
{
    a = a + b;
    return a;
}

```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

19. Si **pgcd** est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`
- ☐ `n = pgcd(int p, int q);`
- ☐ `n = pgcd(n, 3);`
- ☐ `int pgcd(2);`

20. Pour l'extrait de programme suivant :

```

int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);

```

La valeur affichée est :

- ☐ 8
- ☐ 16
- ☐ 0
- ☐ 4

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur
- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur
- ☐ Transférer des données et intructions entre processeur et mémoire

2. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`
- ☐ `int pgcd(2);`
- ☐ `n = pgcd(n, 3);`
- ☐ `n = pgcd(int p, int q);`

3. Un registre du processeur est :

- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ un composant qui contient la liste des fichiers du système

4. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 5
- ☐ 4
- ☐ 1
- ☐ 0

5. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ i
- ☐ A
- ☐ ABCDEF
- ☐ cccccc

6. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `char c;`
- ☐ `int char;`

7. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `begin`
- ☐ `main`
- ☐ `init`
- ☐ `include`

8. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(racine(x)*racine(x));`

9. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, int x);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, y);`

10. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy(p, q);`
- ☐ `x = mccarthy(n);`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy();`

11. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a{n, x} == b{n, x}`
- ☐ `a == b`
- ☐ `a = b`
- ☐ `(a.n == b.n) && (a.x == b.x)`

12. Un fichier source est :

- ☐ un document illisible pour les humains
- ☐ un document qui doit être protégé
- ☐ un document de référence du système
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un fichier texte qui sera traduit en instructions processeur

13. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5

14. L'ordonnancement par tourniquet permet :

- ☐ de doubler la mémoire disponible
- ☐ d'afficher des ronds colorés à l'écran
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'entretenir l'illusion que les processus tournent en parallèle

15. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction main(), alors on peut écrire en début de main() :

- ☐ toto\_s n, x;
- ☐ toto\_s struct z = {3, 0.5};
- ☐ struct toto\_s toto;
- ☐ int struct toto\_s = {3, -1e10};
- ☐ int toto.n = 3;

16. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 15
- ☐ 10
- ☐ 6
- ☐ 0

17. Sous unix (ou linux), la commande ls permet de :

- ☐ compiler un programme
- ☐ voir des clips musicaux
- ☐ afficher le contenu d'un fichier texte
- ☐ afficher la liste de fichiers contenus dans un répertoire

18. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
```

```
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Mineur Majeur
- ☐ rien
- ☐ Majeur

19. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses champs
- ☐ ses blocs
- ☐ ses chants

20. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ int %d;
- ☐ int k;
- ☐ loop i;
- ☐ int loop n;

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0

2. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 8
- ☐ 0
- ☐ 4

3. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `int saisie_utilisateur();`

4. Le bus système sert à :

- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et intructions entre processeur et mémoire

5. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 0
- ☐ 4
- ☐ 16

6. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define taille = N`
- ☐ `#define N = 3`
- ☐ `#define taille = 3`
- ☐ `#define N 3`

7. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1

8. L'ordonnancement par tourniquet permet :

- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'afficher des ronds colorés à l'écran
- ☐ de doubler la mémoire disponible

9. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`

10. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 0.5
- ☐ 1
- ☐ 0
- ☐ 1.5

11. Quel est l'opérateur de différence en C :

- ☐ `<>`
- ☐ `!`
- ☐ `!=`
- ☐ `≠`

12. Le langage C est un langage

- ☐ composé
- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ interprété

13. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int toto.n = 3;`
- ☐ `toto_s n, x;`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `struct toto_s toto;`

14. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses cases
- ☐ ses chants
- ☐ ses blocs

15. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ répéter un bloc tant qu'une condition est vérifiée
- ☐ retourner un bloc
- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ mettre les blocs en séquence les uns à la suite des autres

16. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
}
```

```
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 7

17. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique

18. Au début de la fonction main() on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b
- ☐ B
- ☐ A
- ☐ C

19. Pour déclarer une fonction factorielle qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ int factorielle(double n);
- ☐ int factorielle(int x);
- ☐ int factorielle();
- ☐ struct int factorielle(int n);

20. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 0
- ☐ 1
- ☐ 6
- ☐ 42



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Sous unix (ou linux), la commande `ls` permet de :
  - ☐ afficher le contenu d'un fichier texte
  - ☐ afficher la liste de fichiers contenus dans un répertoire
  - ☐ voir des clips musicaux
  - ☐ compiler un programme
- Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction déclarée mais non définie
  - ☐ une fonction appelée avant sa déclaration
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
- Après exécution jusqu'à la ligne 14 du programme C :
 

```

10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

  - ☐ le terminal affiche "Faux"
  - ☐ le terminal affiche 5
  - ☐ le terminal affiche `x = 2`
  - ☐ le terminal affiche `x = 5`
- Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
  - ☐ `int char;`
  - ☐ `char 'c';`
  - ☐ `char c;`
  - ☐ `char "c";`

- Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?
  - ☐ un point-virgule en trop
  - ☐ une accolade en trop
  - ☐ un point-virgule manquant
  - ☐ une accolade manquante
- Pour compiler un programme `prog.c`, on utilise la ligne de commande :
  - ☐ `gcc -Wall prog.exe -o prog.c`
  - ☐ `gcc -Wall prog.c -o prog.exe`
  - ☐ `gcc prog.c -o -Wall prog.exe`
  - ☐ `gcc prog.exe -Wall -o prog.c`
- Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
  - ☐ `int exposant(double n, int x);`
  - ☐ `void exposant(double x^n);`
  - ☐ `exposant(double x, int n, int r);`
  - ☐ `double exposant(double x, int n);`
- Sur unix (ou linux), la commande `mkdir` permet de :
  - ☐ changer de répertoire courant
  - ☐ ouvrir un fichier texte
  - ☐ créer un répertoire
  - ☐ créer un fichier texte
- Un enregistrement permet de grouper plusieurs valeurs dans :
  - ☐ ses chants
  - ☐ ses blocs
  - ☐ ses cases
  - ☐ ses champs
- Une variable booléenne est un variable :
  - ☐ NaN (not a number, qui n'est pas un nombre)
  - ☐ jamais nulle
  - ☐ réelle positive
  - ☐ qui est vraie ou fausse
  - ☐ à laquelle une valeur vient d'être affectée

- Le code suivant :

```

int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Mineur  
Majeur
- ☐ rien

- Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :
  - ☐ `void saisie_utilisateur(char c);`
  - ☐ `void saisie_utilisateur(int n);`
  - ☐ `int saisie_utilisateur();`
  - ☐ `saisie_utilisateur(scanf(%d));`
- Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :
  - ☐ `double afficher_menu();`
  - ☐ `void afficher_menu();`
  - ☐ `int afficher_menu(int char);`
  - ☐ `int afficher_menu();`
  - ☐ `char afficher_menu(printf("menu"));`
- Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
  - ☐ un débogueur
  - ☐ `printf("Valeur de n ? %d\n", n);`
  - ☐ `printf("Valeur de n ? %g\n", n);`
  - ☐ `scanf("%d", &n);`

15. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

16. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int y, int x);`

17. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou`  
`référence indéfinie vers « printf »`

- ☐ l'édition de liens
- ☐ l'analyse des entrées clavier
- ☐ l'analyse sémantique
- ☐ l'analyse harmonique

18. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ `j = 0`
- ☐ `j = 4`
- ☐ `j = %d`
- ☐ `j = 5`

19. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(int n);`
- ☐ `n = carre(n);`
- ☐ `int carre(2);`
- ☐ `int n = carre();`

20. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`
- ☐ `int pgcd(2);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `int n = pgcd();`

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. L'ordonnancement par tourniquet permet :

- ☐ de doubler la mémoire disponible
- ☐ d'afficher des ronds colorés à l'écran
- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ de ne pas perdre de temps avec la commutation de contexte

2. Sous unix (ou linux), la commande `ls` permet de :

- ☐ compiler un programme
- ☐ voir des clips musicaux
- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ afficher le contenu d'un fichier texte

3. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse syntaxique
- ☐ analyse lexicale
- ☐ analyse sémantique

4. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

5. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`

6. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `A && B`
- ☐ `(!A || B)`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `!(A || B) == (A && !B)`

7. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

8. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ `4 3 2 1 0`
- ☐ `0 1 2 3 4`
- ☐ `4 3 2 1`
- ☐ `0 1 2 3`

9. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`

10. Si cet avertissement apparaît à la compilation :

**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante

11. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

12. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ le programme affiche "Faux"
- ☐ la variable y vaut 5
- ☐ la variable x vaut 0
- ☐ la variable x vaut 5 et la variable y vaut 0

13. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il comporte une boucle infinie

14. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`
- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy(p, q);`
- ☐ `int mccarthy(int 2);`

15. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou référence indéfinie vers << printf >>`

- ☐ l'analyse sémantique
- ☐ l'édition de liens
- ☐ l'analyse des entrées clavier
- ☐ l'analyse harmonique

16. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 0 1 2 3 4 5 6
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6

17. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = 5

- ☐ j = %d
- ☐ j = 0
- ☐ j = 4

18. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1
- ☐ 4
- ☐ 0
- ☐ 5

19. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur
- ☐ qu'on veut changer aléatoirement de fond d'écran

20. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A
- ☐ C
- ☐ B
- ☐ b

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`  
☐ `(n<=a) && (n<=b)`  
☐ `(a<=n) && (n<=b)`  
☐ `(a<n) || (n>b)`

2. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0  
☐ 7  
☐ 5

3. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur  
☐ `printf("Valeur de n ? %g\n", n);`  
☐ `scanf("%d", &n);`  
☐ `printf("Valeur de n ? %d\n", n);`

4. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'il faut lancer un débogueur  
☐ qu'il faut indenter le fichier source  
☐ que l'on veut voir tous les avertissements  
☐ qu'on veut changer aléatoirement de fond d'écran

5. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`  
☐ `n = pgcd(n, 3);`  
☐ `n = pgcd(int p, int q);`  
☐ `int pgcd(2);`

6. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 4  
☐ 5  
☐ 101  
☐ 3

7. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tous ensemble  
☐ en parallèle, chacun dans un registre  
☐ tour à tour, un petit peu à chaque fois  
☐ chacun son tour, après que le processus précédent a terminé

8. Un bit est :

- ☐ la longueur d'un mot mémoire  
☐ un battement d'horloge processeur  
☐ un chiffre binaire (0 ou 1)  
☐ l'instruction qui met fin à un programme

9. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define N 3`  
☐ `#define taille = N`  
☐ `#define N = 3`  
☐ `#define taille = 3`

10. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3  
☐ 16  
☐ 20  
☐ 6

11. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal  
☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ vous n'avez pas déclaré de fonction

12. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2 3  
☐ 0 1 2 0 1 2  
☐ 0 1 2 0 1 2 3  
☐ 0 1 2 3 0 1 2

13. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1
- ☐ 5
- ☐ 0
- ☐ 4

14. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `(!A || B)`
- ☐ `A && B`
- ☐ `!(!A || B) == (A && !B)`

15. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`

16. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7
- ☐ 3
- ☐ 8
- ☐ 111

17. Le type des réels en C est :

- ☐ `int`
- ☐ `double`
- ☐ `real`
- ☐ `char`

18. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs

- ☐ ses chants
- ☐ ses blocs
- ☐ ses cases

19. Le langage C est un langage

- ☐ interprété
- ☐ lu, écrit, parlé
- ☐ compilé
- ☐ composé

20. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si  $x$  est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 0  
☐ 1.5  
☐ 0.5  
☐ 1

2. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2 3  
☐ 0 1 2 0 1 2  
☐ 0 1 2 3 0 1 2  
☐ 0 0 1 1 2 2 3

3. Soit la fonction  $f$  définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression  $f(0)$  prendra la valeur :

- ☐ 4  
☐ 0  
☐ 3

4. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3  
☐ 8  
☐ 7  
☐ 111

5. Soit la fonction  $g$  définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression  $g(0)$  prendra la valeur :

- ☐ 5  
☐ 0  
☐ 7

6. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ i  
☐ ABCDEF  
☐ cccccc  
☐ A

7. Le langage C est un langage

- ☐ lu, écrit, parlé  
☐ composé  
☐ interprété  
☐ compilé

8. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 0 1 1 1  
☐ 1 2 1 2 3  
☐ 0 1 0 1 0 1 0 1  
☐ 0 1 2 0 1 2

9. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`  
☐ `char "c";`  
☐ `char c;`  
☐ `int char;`

10. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`  
☐ `void exposant(double x^n);`  
☐ `double exposant(double x, int n);`  
☐ `int exposant(double n, int x);`

11. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy(p, q);`  
☐ `n = mccarthy();`  
☐ `int mccarthy(int 2);`  
☐ `x = mccarthy(n);`

12. Une variable booléenne est une variable :

- ☐ à laquelle une valeur vient d'être affectée
- ☐ jamais nulle
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ qui est vraie ou fausse
- ☐ réelle positive

13. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#appart <stdlib.h>`
- ☐ `#include <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <stdio.h>`

14. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

15. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(double n);`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`

16. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`
- ☐ `a<=n<=b`
- ☐ `(a<n) || (n>b)`
- ☐ `(n<=a) && (n<=b)`

17. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ `j = 0`
- ☐ `j = 4`
- ☐ `j = %d`
- ☐ `j = 5`

18. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche `x = 2`
- ☐ le terminal affiche "Faux"
- ☐ le terminal affiche 5
- ☐ le terminal affiche `x = 5`

19. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`

20. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 1 2 3 4

2. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.

- ☐ #define N = 3
- ☐ #define taille = 3
- ☐ #define N 3
- ☐ #define taille = N

3. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

4. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche x = 2
- ☐ le terminal affiche 5
- ☐ le terminal affiche "Faux"
- ☐ le terminal affiche x = 5

5. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 1
- ☐ 4
- ☐ 5
- ☐ 0

6. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 8
- ☐ 0
- ☐ 4

7. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`
- ☐ `int mccarthy(int 2);`
- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy(p, q);`

8. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 5
- ☐ j = 4
- ☐ j = %d
- ☐ j = 0

9. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

- ☐ le programme affiche "Faux"
- ☐ la variable y vaut 5
- ☐ la variable x vaut 3
- ☐ la variable x vaut 5 et la variable y vaut 3

10. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien

11. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ i
- ☐ A
- ☐ ABCDEF
- ☐ cccccc

12. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction

13. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée et définie
- ☐ l'avoir déclarée
- ☐ l'avoir définie
- ☐ avoir défini une constante symbolique de la taille de cette fonction

14. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(n);`
- ☐ `int n = carre();`
- ☐ `int carre(2);`
- ☐ `n = carre(int n);`

15. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 16
- ☐ 3
- ☐ 6
- ☐ 20

16. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7
- ☐ 111
- ☐ 3
- ☐ 8

17. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char 'c';`

18. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a == b`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`

19. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%d et y=%d\n",x y);`

20. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable x vaut 0
- ☐ la variable y vaut 5
- ☐ la variable x vaut 5 et la variable y vaut 0
- ☐ le programme affiche "Faux"

2. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 111
- ☐ 7
- ☐ 3
- ☐ 8

3. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6 8
- ☐ 0 2 4 6

4. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'on veut changer aléatoirement de fond d'écran

5. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1

6. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ rien

7. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 3
- ☐ 4

8. Une variable booléenne est un variable :

- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ qui est vraie ou fausse
- ☐ à laquelle une valeur vient d'être affectée
- ☐ réelle positive
- ☐ jamais nulle

9. Si cette erreur apparaît à la compilation :

**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ une variable non déclarée
- ☐ un caractère interdit en C
- ☐ une faute de frappe dans un appel de fonction

10. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a == b`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`

11. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle `[a..b]`, on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ a<=n<=b
- ☐ (a<n) || (n>b)
- ☐ (n<=a) && (n<=b)
- ☐ (a<=n) && (n<=b)

12. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction

13. Un bit est :

- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur
- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme

14. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3
- ☐ 4 3 2 1

15. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ (A == TRUE) && (B == TRUE)
- ☐ A && B
- ☐ (!(A || B) == (A && !B))
- ☐ (!A || B)

16. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
- ☐ 0 0 1 1 2 2
- ☐ 1 2 3 1 2

17. Lorsqu'un programme utilise printf ou scanf il faut qu'il contienne l'instruction préprocesseur :

- ☐ #appart <stdlib.h>
- ☐ #include <studio.h>
- ☐ #include <studlib.h>
- ☐ #include <stdio.h>

18. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
```

```
int b = 5;
printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5

19. Pour déclarer une fonction factorielle qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ int factorielle();
- ☐ struct int factorielle(int n);
- ☐ int factorielle(int x);
- ☐ int factorielle(double n);

20. Au début de la fonction main() on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A
- ☐ C
- ☐ B
- ☐ b

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 0  
☐ j = 4  
☐ j = 5  
☐ j = %d

2. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a == b  
☐ a = b  
☐ (a.n == b.n) && (a.x == b.x)  
☐ a{n, x} == b{n, x}

3. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ double exposant(double x, int n);  
☐ exposant(double x, int n, int r);  
☐ void exposant(double x^n);  
☐ int exposant(double n, int x);

4. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ main  
☐ init  
☐ include  
☐ begin

5. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants  
☐ ses blocs  
☐ ses champs  
☐ ses cases

6. Le type des réels en C est :

- ☐ real  
☐ char  
☐ int  
☐ double

7. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction déclarée mais non définie  
☐ une directive préprocesseur **#include** manquante  
☐ une fonction appelée avant sa déclaration

8. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 5  
☐ 0  
☐ 1  
☐ 4

9. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque  
☐ dans lequel vous avez déclaré ces fonction  
☐ alphabétique  
☐ dans lequel ces fonctions sont appelées dans le main

10. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction **main()**, alors on peut écrire en début de **main()** :

- ☐ int toto.n = 3;  
☐ toto\_s n, x;  
☐ int struct toto\_s = {3, -1e10};  
☐ toto\_s struct z = {3, 0.5};  
☐ struct toto\_s toto;

11. Après la déclaration : **int mccarthy(int n);**, il est correct d'écrire :

- ☐ n = mccarthy();  
☐ n = mccarthy(p, q);  
☐ x = mccarthy(n);  
☐ int mccarthy(int 2);

12. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tour à tour, un petit peu à chaque fois  
☐ chacun son tour, après que le processus précédent a terminé  
☐ tous ensemble  
☐ en parallèle, chacun dans un registre

13. Si cette erreur apparaît à la compilation :  
`Undefined symbols : "_printf" ou`  
`référence indéfinie vers « printf »` que doit-on chercher dans le programme ?

- ☐ un caractère interdit en C
- ☐ une faute de frappe dans un appel de fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une variable non déclarée

14. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 0 2 4 6
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 1 2 3 4 5 6

15. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 0
- ☐ 5

16. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 0
- ☐ 16
- ☐ 4

17. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 6
- ☐ 42
- ☐ 1
- ☐ 0

18. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(double n);`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`

19. Un bit est :

- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur
- ☐ la longueur d'un mot mémoire

20. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 3
- ☐ 4

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur

2. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ cccccc
- ☐ A
- ☐ ABCDEF
- ☐ i

3. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 0
- ☐ 6
- ☐ 10
- ☐ 15

4. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2
- ☐ 8 6 4 2 0
- ☐ 8 2
- ☐ 0 2 4 6 8

5. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration

6. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

7. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<n) || (n>b)`

8. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir définie
- ☐ l'avoir déclarée et définie
- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir déclarée

9. Vous utilisez une boucle `while` quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

10. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ les fichiers du disque
- ☐ des processus
- ☐ en temps d'accès
- ☐ certaines données de la mémoire de travail

11. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=8, a=3, b=5

12. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée

13. Pour déclarer une fonction pgcd qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ int pgcd(int x, int x);
- ☐ int pgcd(int y, int x);
- ☐ void pgcd(int x, int y);
- ☐ int pgcd(int x, y);

14. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 0 1 2 3
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1

15. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0

16. Si cette erreur apparaît à la compilation :

**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop
- ☐ une accolade manquante
- ☐ un point-virgule manquant
- ☐ une accolade en trop

17. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a = b
- ☐ a{n, x} == b{n, x}
- ☐ a == b
- ☐ (a.n == b.n) && (a.x == b.x)

18. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ init
- ☐ include
- ☐ main
- ☐ begin

19. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 0
- ☐ 8
- ☐ 4

20. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque
- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. L'ordonnancement par tourniquet permet :

- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ de doubler la mémoire disponible
- ☐ d'afficher des ronds colorés à l'écran

2. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

3. Le langage C est un langage

- ☐ composé
- ☐ compilé
- ☐ interprété
- ☐ lu, écrit, parlé

4. Un registre du processeur est :

- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ un composant qui contient la liste des fichiers du système

5. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(2/3);`
- ☐ `x - 1 = racine(x);`

6. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 1.5
- ☐ 1
- ☐ 0
- ☐ 0.5

7. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`  
, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur `#include` manquante

8. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

- ☐ la variable `x` vaut 5 et la variable `y` vaut 3
- ☐ la variable `y` vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable `x` vaut 3

9. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int toto.n = 3;`
- ☐ `struct toto_s toto;`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `toto_s n, x;`
- ☐ `toto_s struct z = {3, 0.5};`

10. Le type des réels en C est :

- ☐ `int`
- ☐ `real`
- ☐ `double`
- ☐ `char`

11. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int y, int x);`
- ☐ `void pgcd(int x, int y);`

12. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur

- ☐ rien
- ☐ Mineur
- ☐ Mineur  
Majeur

13. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur **#include** manquante

14. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 0 1 2 3
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0

15. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 5
- ☐ j = %d
- ☐ j = 4
- ☐ j = 0

16. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui ne sera pas détectée
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ ne comportent aucune erreur

17. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`

18. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses cases
- ☐ ses champs
- ☐ ses chants

19. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char c;`
- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `int char;`

20. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `begin`
- ☐ `init`
- ☐ `include`
- ☐ `main`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

☐ `int afficher_menu();`  
☐ `void afficher_menu();`  
☐ `char afficher_menu(sprintf("menu"));`  
☐ `double afficher_menu();`  
☐ `int afficher_menu(int char);`

2. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

☐ `!(A || B) == (A && !B)`  
☐ `A && B`  
☐ `(A == TRUE) && (B == TRUE)`  
☐ `(!A || B)`

3. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

☐ 0  
☐ 10  
☐ 6  
☐ 15

4. Après exécution jusqu'à la ligne 15 du programme C :

```
10 ...
11 int main() {
12     int x = 5;
13
14     x = 3 * x + 1;
15
16     ...
17 }
```

☐ le programme affiche x  
☐ la variable x vaut  $-\frac{1}{2}$   
☐ la variable x vaut 16  
☐ le programme affiche \*\*\*\*

5. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

☐ 0 2 4 6  
☐ 0 1 2 3 4 5 6 7  
☐ 0 2 4 6 8  
☐ 0 1 2 3 4 5 6

6. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

☐ `x = racine(racine(x)*racine(x));`  
☐ `x = racine(2/3);`  
☐ `x - 1 = racine(x);`  
☐ `x = racine(x * x) - racine(x);`

7. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

☐ `void afficher_date(struct date_s d);`  
☐ `int afficher_date(date_s d);`  
☐ `void afficher_date(date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`

8. L'écriture 111 en binaire correspond au nombre naturel :

☐ 111  
☐ 3  
☐ 8  
☐ 7

9. Un registre du processeur est :

☐ une unité de calcul spécialisée de l'ordinateur  
☐ un composant qui contient la liste des fichiers du système  
☐ une gamme de fréquence de fonctionnement du processeur  
☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

10. Un fichier source est :

☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur  
☐ un document de référence du système  
☐ un fichier texte qui sera traduit en instructions processeur  
☐ un document qui doit être protégé  
☐ un document illisible pour les humains

11. Le bus système sert à :

☐ Écrire des données sur le disque dur  
☐ Transférer des données et intructions entre processeur et mémoire  
☐ transporter les processus du tourniquet au processeur  
☐ Arriver à l'heure en cours

12. Le langage C est un langage

☐ composé  
☐ interprété  
☐ lu, écrit, parlé  
☐ compilé

13. Un programme en langage C doit comporter une et une seule définition de la fonction :

☐ include  
☐ begin  
☐ main  
☐ init

14. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses blocs
- ☐ ses cases
- ☐ ses champs

15. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 20
- ☐ 6
- ☐ 3
- ☐ 16

16. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction main(), alors on peut écrire en début de main() :

- ☐ toto\_s n, x;
- ☐ struct toto\_s toto;
- ☐ toto\_s struct z = {3, 0.5};
- ☐ int struct toto\_s = {3, -1e10};
- ☐ int toto.n = 3;

17. Après la déclaration : int mccarthy(int n);, il est correct d'écrire :

- ☐ n = mccarthy(p, q);
- ☐ int mccarthy(int 2);
- ☐ n = mccarthy();
- ☐ x = mccarthy(n);

18. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ scanf("%d", &n);
- ☐ un débogueur
- ☐ printf("Valeur de n ? %d\n", n);
- ☐ printf("Valeur de n ? %g\n", n);

19. Pour compiler un programme prog.c, on utilise la ligne de commande :

- ☐ gcc prog.c -o -Wall prog.exe
- ☐ gcc -Wall prog.c -o prog.exe
- ☐ gcc -Wall prog.exe -o prog.c
- ☐ gcc prog.exe -Wall -o prog.c

20. Pour déclarer une fonction exposant qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ exposant(double x, int n, int r);
- ☐ double exposant(double x, int n);
- ☐ void exposant(double x^n);
- ☐ int exposant(double n, int x);

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4  
☐ 3  
☐ 0

2. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4  
☐ 4 3 2 1  
☐ 0 1 2 3 4  
☐ 4 3 2 1 0

3. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max', que doit-on chercher dans le programme ?`

- ☐ une fonction appelée avant sa déclaration  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction déclarée mais non définie  
☐ une directive préprocesseur `#include` manquante

4. Sous unix (ou linux), la commande `ls` permet de :

- ☐ compiler un programme  
☐ afficher la liste de fichiers contenus dans un répertoire  
☐ voir des clips musicaux  
☐ afficher le contenu d'un fichier texte

5. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu(int char);`  
☐ `int afficher_menu();`  
☐ `double afficher_menu();`  
☐ `void afficher_menu();`  
☐ `char afficher_menu(printf("menu"));`

6. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y = 3;
13
14     x = y;
15
16     ...
17 }
```

- ☐ le programme affiche "Faux"  
☐ la variable `x` vaut 5 et la variable `y` vaut 3  
☐ la variable `y` vaut 5  
☐ la variable `x` vaut 3

7. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %g\n", n);`  
☐ un débogueur  
☐ `printf("Valeur de n ? %d\n", n);`  
☐ `scanf("%d", &n);`

8. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir définie  
☐ l'avoir déclarée et définie  
☐ l'avoir déclarée  
☐ avoir défini une constante symbolique de la taille de cette fonction

9. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 5  
☐ 4  
☐ 1  
☐ 0

10. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<=5;i=i+1)`  
☐ `for(i=0;i<5;i=i+1)`  
☐ `for(i=1;i<5;i=i+1)`  
☐ `for(i=1;i<=5;i=i+1)`

11. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

- ☐ 0  
☐ 0.5  
☐ 1  
☐ 1.5

12. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 0
- ☐ 5

13. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements

14. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3
- ☐ 6
- ☐ 20
- ☐ 16

15. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc -Wall prog.c -o prog.exe`

16. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`

17. Un fichier source est :

- ☐ un document de référence du système
- ☐ un document illisible pour les humains
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document qui doit être protégé
- ☐ un fichier texte qui sera traduit en instructions processeur

18. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ la division du programme en zones homogènes échoue
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal

19. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`
- ☐ `int mccarthy(int 2);`
- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy(p, q);`

20. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant
- ☐ un point-virgule en trop
- ☐ une accolade manquante
- ☐ une accolade en trop

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(racine(x)*racine(x));`

2. Pour déclarer une fonction **pgcd** qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, int x);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, y);`

3. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque
- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le main

4. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`
- ☐ `int factorielle(double n);`

5. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche `x = 5`
- ☐ le terminal affiche `x = 2`
- ☐ le terminal affiche 5
- ☐ le terminal affiche "Faux"

6. Au début de la fonction **main()** on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ cccccc
- ☐ i
- ☐ A

7. Pour déclarer une fonction **exposant** qui prend en argument un réel *x* et un entier positif *n* et renvoie la valeur de  $x^n$  on écrit :

- ☐ `double exposant(double x, int n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`
- ☐ `void exposant(double x^n);`

8. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 4
- ☐ 16
- ☐ 0
- ☐ 8

9. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses blocs
- ☐ ses chants
- ☐ ses cases

10. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou

11. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable `x` vaut 5 et la variable `y` vaut 0
- ☐ la variable `y` vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable `x` vaut 0

12. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

Undefined symbols : "\_printf" ou  
référence indéfinie vers < printf >

- ☐ l'analyse sémantique
- ☐ l'analyse des entrées clavier
- ☐ l'analyse harmonique
- ☐ l'édition de liens

13. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction
- ☐ `loop i;`
  - ☐ `int %d;`
  - ☐ `int loop n;`
  - ☐ `int k;`
14. Le code suivant :
- ```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```
- affichera :
- ☐ Mineur
  - ☐ Majeur
  - ☐ rien
  - ☐ Majeur
  - ☐ Mineur
15. Vous utilisez une boucle `while` quand :
- ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

- ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ vous n'avez pas déclaré de fonction
16. L'écriture 101 en binaire correspond au nombre naturel :
- ☐ 3
  - ☐ 4
  - ☐ 5
  - ☐ 101
17. Après exécution du programme :
- ```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```
- ☐ le terminal affiche 8
  - ☐ la case mémoire 8 contiendra 0
  - ☐ le bus explose
  - ☐ la case mémoire 8 contiendra 16
18. Un bit est :
- ☐ un battement d'horloge processeur
  - ☐ l'instruction qui met fin à un programme
  - ☐ un chiffre binaire (0 ou 1)
  - ☐ la longueur d'un mot mémoire

19. Le langage C est un langage
- ☐ lu, écrit, parlé
  - ☐ interprété
  - ☐ compilé
  - ☐ composé
20. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :
- ```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```
- Quelle est la condition `cond` :
- ☐ `(a<n) || (n>b)`
  - ☐ `a<=n<=b`
  - ☐ `(n<=a) && (n<=b)`
  - ☐ `(a<=n) && (n<=b)`



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3  
☐ 0  
☐ 4

2. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs  
☐ ses blocs  
☐ ses chants  
☐ ses cases

3. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`  
☐ `a<=n<=b`  
☐ `(n<=a) && (n<=b)`  
☐ `(a<=n) && (n<=b)`

4. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire  
☐ Écrire des données sur le disque dur  
☐ Arriver à l'heure en cours  
☐ transporter les processus du tourniquet au processeur

5. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`  
☐ `int pgcd(int x, int x);`  
☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int y, int x);`

6. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ `** ** *`  
☐ `*****`  
☐ `**** *`  
☐ `** ***`

7. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define taille = N`  
☐ `#define N 3`  
☐ `#define taille = 3`  
☐ `#define N = 3`

8. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C  
☐ B  
☐ b  
☐ A

9. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`  
☐ `x = mccarthy(n);`  
☐ `int mccarthy(int 2);`  
☐ `n = mccarthy(p, q);`

10. Le langage C est un langage

- ☐ compilé  
☐ lu, écrit, parlé  
☐ composé  
☐ interprété

11. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée  
☐ certains programmes sont de vrais plats de spaghetti  
☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine  
☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine

12. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc prog.c -o -Wall prog.exe`  
☐ `gcc prog.exe -Wall -o prog.c`  
☐ `gcc -Wall prog.c -o prog.exe`  
☐ `gcc -Wall prog.exe -o prog.c`

13. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »**
- ☐ l'analyse des entrées clavier
  - ☐ l'analyse harmonique
  - ☐ l'analyse sémantique
  - ☐ l'édition de liens
14. L'ordonnancement par tourniquet permet :
- ☐ d'afficher des ronds colorés à l'écran
  - ☐ de doubler la mémoire disponible
  - ☐ de ne pas perdre de temps avec la commutation de contexte
  - ☐ d'entretenir l'illusion que les processus tournent en parallèle
15. Si  $x$  est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :
- ☐ 0
  - ☐ 1
  - ☐ 1.5
  - ☐ 0.5

16. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
- ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une directive préprocesseur **#include** manquante
  - ☐ une fonction déclarée mais non définie
  - ☐ une fonction appelée avant sa déclaration
17. Quel est le problème d'un programme comportant les lignes suivantes ?
- ```
while (1)
{
    printf("coucou\n");
}
```
- ☐ il ne compile pas
  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il comporte une boucle infinie
  - ☐ il n'affiche rien
18. Si **pgcd** est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :
- ☐ **int pgcd(2);**

- ☐ **n = pgcd(int p, int q);**
  - ☐ **n = pgcd(n, 3);**
  - ☐ **int n = pgcd();**
19. Quel est l'opérateur de différence en C :
- ☐ **≠**
  - ☐ **!=**
  - ☐ **!**
  - ☐ **<>**
20. Après exécution du programme :
- ```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```
- ☐ la case mémoire 8 contiendra 0
  - ☐ le bus explose
  - ☐ le terminal affiche 8
  - ☐ la case mémoire 8 contiendra 16

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut lancer un débogueur

2. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction

3. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`

4. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 2
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2 0
- ☐ 8 6 4 2

5. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0

6. Une variable booléenne est un variable :

- ☐ qui est vraie ou fausse
- ☐ à laquelle une valeur vient d'être affectée
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ jamais nulle
- ☐ réelle positive

7. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ la variable `x` vaut 0
- ☐ la variable `x` vaut 5 et la variable `y` vaut 0
- ☐ la variable `y` vaut 5
- ☐ le programme affiche "Faux"

8. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction

9. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ un débogueur
- ☐ `scanf("%d", &n);`

10. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc -Wall prog.exe -o prog.c`

11. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `!(A || B) == (A && !B)`
- ☐ `(!A || B)`
- ☐ `A && B`

12. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses cases
- ☐ ses champs
- ☐ ses chants

13. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou référence indéfinie vers « printf »`
- ☐ l'analyse harmonique
  - ☐ l'édition de liens
  - ☐ l'analyse sémantique
  - ☐ l'analyse des entrées clavier
14. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
- ☐ `void exposant(double x^n);`
  - ☐ `exposant(double x, int n, int r);`
  - ☐ `int exposant(double n, int x);`
  - ☐ `double exposant(double x, int n);`
15. Le type des réels en C est :
- ☐ `char`
  - ☐ `real`
  - ☐ `double`
  - ☐ `int`
16. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
- ☐ `char "c";`
  - ☐ `char 'c';`
  - ☐ `char c;`
  - ☐ `int char;`
17. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
- ☐ `int pgcd(int y, int x);`
  - ☐ `int pgcd(int x, y);`
  - ☐ `void pgcd(int x, int y);`
  - ☐ `int pgcd(int x, int x);`
18. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
- ☐ `int factorielle(int 2);`
  - ☐ `n = factorielle();`
  - ☐ `printf("%d", factorielle(n));`
  - ☐ `n = factorielle(p, q);`
19. L'écriture 101 en binaire correspond au nombre naturel :
- ☐ 3
  - ☐ 5
  - ☐ 101
  - ☐ 4
20. Le code suivant :
- ```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 4 3 2 1 0
  - ☐ 1 2 3 4
  - ☐ 0 1 2 3 4
  - ☐ 4 3 2 1

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- Une *segmentation fault* est une erreur qui survient lorsque :
  - ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
  - ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
  - ☐ la division du programme en zones homogènes échoue
  - ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- Quel est le problème d'un programme comportant les lignes suivantes ?
 

```
while (1)
{
    printf("coucou\n");
}
```

  - ☐ il comporte une boucle infinie
  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il ne compile pas
  - ☐ il n'affiche rien
- Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction
  - ☐ `int toto[taille=5];`
  - ☐ `int toto[5];`
  - ☐ `char tableau[5];`
  - ☐ `int[] new tableau(5);`
  - ☐ `int tab[] = 5;`
- Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?
  - ☐ un point-virgule manquant
  - ☐ un point-virgule en trop
  - ☐ une accolade en trop
  - ☐ une accolade manquante

- Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :
  - ☐ `struct int factorielle(int n);`
  - ☐ `int factorielle(int x);`
  - ☐ `int factorielle();`
  - ☐ `int factorielle(double n);`
- Le code suivant :
 

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

 affichera :
  - ☐ rien
  - ☐ Mineur
  - ☐ Majeur
  - ☐ Mineur
  - ☐ Majeur
- Si *a* et *b* sont deux variables de type :
 

```
struct toto_s
{
    int n;
    double x;
};
```

 Alors pour tester l'égalité de *a* et de *b* on utilise la condition :
  - ☐ `a = b`
  - ☐ `(a.n == b.n) && (a.x == b.x)`
  - ☐ `a == b`
  - ☐ `a{n, x} == b{n, x}`

- On souhaite faire une boucle de contrôle de saisie : tant que l'entier *n* n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de *n*. Soit le programme suivant :
 

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

 Quelle est la condition *cond* :
  - ☐ `(n<=a) && (n<=b)`
  - ☐ `a<=n<=b`
  - ☐ `(a<n) || (n>b)`
  - ☐ `(a<=n) && (n<=b)`
- Si **factorielle** est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
  - ☐ `n = factorielle(p, q);`
  - ☐ `int factorielle(int 2);`
  - ☐ `n = factorielle();`
  - ☐ `printf("%d", factorielle(n));`
- Si *n* est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
  - ☐ un débogueur
  - ☐ `printf("Valeur de n ? %d\n", n);`
  - ☐ `scanf("%d", &n);`
  - ☐ `printf("Valeur de n ? %g\n", n);`
- Lorsqu'un programme utilise **printf** ou **scanf** il faut qu'il contienne l'instruction préprocesseur :
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <stdio.h>`
  - ☐ `#include <studio.h>`
  - ☐ `#include <studlib.h>`
- Le type des réels en C est :
  - ☐ `double`
  - ☐ `char`
  - ☐ `real`
  - ☐ `int`

13. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Majeur
- ☐ rien
- ☐ Mineur

14. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 20
- ☐ 3

☐ 16

☐ 6

15. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

16. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale
- ☐ analyse syntaxique
- ☐ analyse sémantique
- ☐ analyse harmonique

17. Pour compiler un programme prog.c, on utilise la ligne de commande :

- ☐ gcc -Wall prog.c -o prog.exe
- ☐ gcc prog.c -o -Wall prog.exe
- ☐ gcc prog.exe -Wall -o prog.c
- ☐ gcc -Wall prog.exe -o prog.c

18. Une variable booléenne est un variable :

- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ à laquelle une valeur vient d'être affectée
- ☐ qui est vraie ou fausse
- ☐ réelle positive
- ☐ jamais nulle

19. Vous utilisez une boucle while quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un for dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1

20. Pour déclarer une fonction pgcd qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ int pgcd(int x, y);
- ☐ int pgcd(int x, int x);
- ☐ void pgcd(int x, int y);
- ☐ int pgcd(int y, int x);

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- Avant de faire appel à une fonction il est nécessaire de :
  - ☐ avoir défini une constante symbolique de la taille de cette fonction
  - ☐ l'avoir déclarée et définie
  - ☐ l'avoir déclarée
  - ☐ l'avoir définie
- Dans la commande gcc, l'option `-Wall` signifie :
  - ☐ qu'il faut indenter le fichier source
  - ☐ qu'il faut lancer un débogueur
  - ☐ que l'on veut voir tous les avertissements
  - ☐ qu'on veut changer aléatoirement de fond d'écran
- Le langage C est un langage
  - ☐ compilé
  - ☐ lu, écrit, parlé
  - ☐ interprété
  - ☐ composé
- Un registre du processeur est :
  - ☐ une unité de calcul spécialisée de l'ordinateur
  - ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
  - ☐ une gamme de fréquence de fonctionnement du processeur
  - ☐ un composant qui contient la liste des fichiers du système
- Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci : `Undefined symbols : "_printf" ou référence indéfinie vers « printf »`
  - ☐ l'édition de liens
  - ☐ l'analyse des entrées clavier
  - ☐ l'analyse harmonique
  - ☐ l'analyse sémantique

- Une variable booléenne est un variable :
  - ☐ qui est vraie ou fausse
  - ☐ réelle positive
  - ☐ NaN (not a number, qui n'est pas un nombre)
  - ☐ à laquelle une valeur vient d'être affectée
  - ☐ jamais nulle
- L'écriture `111` en binaire correspond au nombre naturel :
  - ☐ 8
  - ☐ 7
  - ☐ 3
  - ☐ 111
- Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :
  - ☐ `n = pgcd(n, 3);`
  - ☐ `int pgcd(2);`
  - ☐ `n = pgcd(int p, int q);`
  - ☐ `int n = pgcd();`
- Soit la fonction `f` définie par :
 

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

 Alors l'expression `f(0)` prendra la valeur :
  - ☐ 3
  - ☐ 4
  - ☐ 0
- Si cette erreur apparaît à la compilation : `error: expected ';' before '}' token` que doit-on chercher dans le programme ?
  - ☐ un point-virgule manquant
  - ☐ une accolade en trop
  - ☐ un point-virgule en trop
  - ☐ une accolade manquante

- Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 16
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose

- On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<n) || (n>b)`

- Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%x et y=%y\n");`

14. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

☐ `void saisie_utilisateur(char c);`  
☐ `void saisie_utilisateur(int n);`  
☐ `saisie_utilisateur(scanf(%d));`  
☐ `int saisie_utilisateur();`

15. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

☐ `n = mccarthy();`  
☐ `n = mccarthy(p, q);`  
☐ `x = mccarthy(n);`  
☐ `int mccarthy(int 2);`

16. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

☐ 5  
☐ 7  
☐ 0

17. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

☐ `(a.n == b.n) && (a.x == b.x)`  
☐ `a == b`  
☐ `a{n, x} == b{n, x}`  
☐ `a = b`

18. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

☐ `A && B`  
☐ `!(A || B) == (A && !B)`  
☐ `(A == TRUE) && (B == TRUE)`  
☐ `(!A || B)`

19. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

☐ `char 'c';`  
☐ `char c;`  
☐ `char "c";`  
☐ `int char;`

20. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

☐ 1 2 3 1 2  
☐ 0 0 1 1 2 2  
☐ 0 1 2 0 1 2  
☐ 0 1 0 1 0 1



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 4
- ☐ 0
- ☐ 16

2. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ B
- ☐ A
- ☐ b
- ☐ C

3. Un bit est :

- ☐ un battement d'horloge processeur
- ☐ un chiffre binaire (0 ou 1)
- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme

4. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
```

```
}
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 4
- ☐ j = %d
- ☐ j = 5
- ☐ j = 0

5. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique

6. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie

7. Le type des réels en C est :

- ☐ double
- ☐ real
- ☐ char
- ☐ int

8. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur
- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur
- ☐ Transférer des données et intructions entre processeur et mémoire

9. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ en temps d'accès
- ☐ certaines données de la mémoire de travail
- ☐ les fichiers du disque
- ☐ des processus

10. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ en parallèle, chacun dans un registre
- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ tous ensemble
- ☐ tour à tour, un petit peu à chaque fois

11. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ rien
- ☐ Mineur  
Majeur

12. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle(p, q);`
- ☐ `printf("%d", factorielle(n));`
- ☐ `int factorielle(int 2);`
- ☐ `n = factorielle();`

13. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a == b
- ☐ a = b
- ☐ a{n, x} == b{n, x}

14. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien
- ☐ il ne compile pas

15. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5

16. Dans la commande gcc, l'option -Wall signifie :

- ☐ qu'il faut lancer un débogueur
- ☐ qu'il faut indenter le fichier source
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'on veut changer aléatoirement de fond d'écran

17. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose
- ☐ la case mémoire 8 contiendra 16

18. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
```

```
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche 5
- ☐ le terminal affiche "Faux"
- ☐ le terminal affiche x = 2
- ☐ le terminal affiche x = 5

19. Pour déclarer une fonction pgcd qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ int pgcd(int x, int x);
- ☐ void pgcd(int x, int y);
- ☐ int pgcd(int x, y);
- ☐ int pgcd(int y, int x);

20. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 5
- ☐ 1
- ☐ 0
- ☐ 4

**Barème : 1 point par réponse juste (unique) ; -0,5 point par réponse fausse. Durée : 20 minutes.**

1. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`  
☐ `n = carre(int n);`  
☐ `n = carre(n);`  
☐ `int n = carre();`

2. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`  
☐ `a<=n<=b`  
☐ `(a<n) || (n>b)`  
☐ `(n<=a) && (n<=b)`

3. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `int factorielle(int 2);`  
☐ `n = factorielle();`  
☐ `printf("%d", factorielle(n));`  
☐ `n = factorielle(p, q);`

4. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
```

```
}
printf("%d", somme);
```

affichera :

- ☐ 1  
☐ 0  
☐ 6  
☐ 42

5. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source  
☐ qu'on veut changer aléatoirement de fond d'écran  
☐ que l'on veut voir tous les avertissements  
☐ qu'il faut lancer un débogueur

6. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop  
☐ un point-virgule manquant  
☐ une accolade en trop  
☐ une accolade manquante

7. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 0.5  
☐ 1  
☐ 0  
☐ 1.5

8. Un registre du processeur est :

- ☐ une unité de calcul spécialisée de l'ordinateur  
☐ un composant qui contient la liste des fichiers du système  
☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs  
☐ une gamme de fréquence de fonctionnement du processeur

9. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `double afficher_menu();`  
☐ `char afficher_menu(printf("menu"));`  
☐ `int afficher_menu(int char);`  
☐ `int afficher_menu();`  
☐ `void afficher_menu();`

10. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque  
☐ dans lequel vous avez déclaré ces fonction  
☐ dans lequel ces fonctions sont appelées dans le `main`  
☐ alphabétique

11. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction  
☐ vous avez déjà fait un `for` dans le même programme principal  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ l'incrément de la variable de boucle n'est pas 1

12. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine  
☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine  
☐ certains programmes sont de vrais plats de spaghetti  
☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée

13. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 4
- ☐ 3

14. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur

15. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3
- ☐ 6
- ☐ 16
- ☐ 20

16. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 0
- ☐ j = 5
- ☐ j = %d
- ☐ j = 4

17. Le langage C est un langage

- ☐ interprété
- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ composé

18. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `void exposant(double x^n);`

19. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ les fichiers du disque
- ☐ certaines données de la mémoire de travail
- ☐ en temps d'accès
- ☐ des processus

20. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur
- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `printf("Valeur de n ? %d\n", n);`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main

2. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`
- ☐ `n = mccarthy(p, q);`
- ☐ `int mccarthy(int 2);`
- ☐ `x = mccarthy(n);`

3. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

4. Un bit est :

- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur

5. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.c -o -Wall prog.exe`

6. Le langage C est un langage

- ☐ composé
- ☐ interprété
- ☐ lu, écrit, parlé
- ☐ compilé

7. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char "c";`

8. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur

9. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ `j = 4`
- ☐ `j = 0`
- ☐ `j = %d`
- ☐ `j = 5`

10. Une variable booléenne est un variable :

- ☐ qui est vraie ou fausse
- ☐ à laquelle une valeur vient d'être affectée
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ jamais nulle
- ☐ réelle positive

11. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 6
- ☐ 42
- ☐ 0
- ☐ 1

12. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a == b`
- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`
- ☐ `(a.n == b.n) && (a.x == b.x)`

13. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?
- ☐ `!(A || B) == (A && !B)`
  - ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `A && B`
  - ☐ `(!A || B)`
14. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction
- ☐ `int tab[] = 5;`
  - ☐ `int toto[taille=5];`
  - ☐ `int toto[5];`
  - ☐ `int[] new tableau(5);`
  - ☐ `char tableau[5];`
15. Après exécution du programme :
- ```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```
- ☐ le bus explose
  - ☐ la case mémoire 8 contiendra 0
  - ☐ le terminal affiche 8
  - ☐ la case mémoire 8 contiendra 16
16. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
- ☐ un débogueur
  - ☐ `scanf("%d", &n);`
  - ☐ `printf("Valeur de n ? %d\n", n);`
  - ☐ `printf("Valeur de n ? %g\n", n);`

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Majeur
- ☐ Mineur Majeur
- ☐ rien
- ☐ Mineur

18. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Mineur Majeur
- ☐ Majeur
- ☐ rien

19. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
- ☐ `(a<n) || (n>b)`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`

20. Soit la fonction  $g$  définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression  $g(0)$  prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ (!A || B)  
☐ (!(A || B) == (A && !B))  
☐ A && B  
☐ (A == TRUE) && (B == TRUE)

2. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction  
☐ un ordre quelconque  
☐ alphabétique  
☐ dans lequel ces fonctions sont appelées dans le main

3. Le langage C est un langage

- ☐ composé  
☐ compilé  
☐ interprété  
☐ lu, écrit, parlé

4. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char 'c';  
☐ char c;  
☐ int char;  
☐ char "c";

5. On souhaite faire une boucle de contrôle de saisie : tant que l'entier n n'appartient pas à l'intervalle [a..b], on recommence la saisie de n. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
```

```
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition cond :

- ☐ (a<n) || (n>b)  
☐ (n<=a) && (n<=b)  
☐ a<=n<=b  
☐ (a<=n) && (n<=b)

6. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ en parallèle, chacun dans un registre  
☐ tour à tour, un petit peu à chaque fois  
☐ chacun son tour, après que le processus précédent a terminé  
☐ tous ensemble

7. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`  
☐ `void saisie_utilisateur(char c);`  
☐ `int saisie_utilisateur();`  
☐ `void saisie_utilisateur(int n);`

8. L'ordonnancement par tourniquet permet :

- ☐ d'entretenir l'illusion que les processus tournent en parallèle  
☐ de ne pas perdre de temps avec la commutation de contexte  
☐ d'afficher des ronds colorés à l'écran  
☐ de doubler la mémoire disponible

9. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
```

```
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4  
☐ 3  
☐ 0

10. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien  
☐ Majeur  
☐ Mineur  
☐ Majeur  
☐ Mineur

11. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`  
☐ `struct toto_s toto;`  
☐ `toto_s struct z = {3, 0.5};`  
☐ `int toto.n = 3;`  
☐ `toto_s n, x;`

12. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?
- ☐ une fonction appelée avant sa déclaration
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction déclarée mais non définie
  - ☐ une directive préprocesseur **#include** manquante

13. L'écriture 101 en binaire correspond au nombre naturel :
- ☐ 4
  - ☐ 3
  - ☐ 5
  - ☐ 101

14. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

15. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?
- ☐ une directive préprocesseur **#include** manquante

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration

16. Lorsqu'un programme utilise **printf** ou **scanf** il faut qu'il contienne l'instruction préprocesseur :

- ☐ **#include <stdio.h>**
- ☐ **#include <studio.h>**
- ☐ **#appart <stdlib.h>**
- ☐ **#include <studlib.h>**

17. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 1
- ☐ 0
- ☐ 42
- ☐ 6

18. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ **int tab[] = 5;**
- ☐ **int toto[taille=5];**
- ☐ **char tableau[5];**
- ☐ **int toto[5];**
- ☐ **int[] new tableau(5);**

19. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6 8
- ☐ 0 2 4 6
- ☐ 0 1 2 3 4 5 6

20. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ cccccc
- ☐ ABCDEF
- ☐ A
- ☐ i

2. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 0
- ☐ 6
- ☐ 15
- ☐ 10

3. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1

4. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ des processus
- ☐ certaines données de la mémoire de travail
- ☐ les fichiers du disque
- ☐ en temps d'accès

5. Le bus système sert à :

- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et instructions entre processeur et mémoire

6. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`

7. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

8. Le type des réels en C est :

- ☐ `double`
- ☐ `real`
- ☐ `char`
- ☐ `int`

9. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`

10. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 111
- ☐ 8
- ☐ 7

11. Si cette erreur apparaît à la compilation : `error: expected ';' before '}' token` que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop
- ☐ un point-virgule manquant
- ☐ une accolade en trop
- ☐ une accolade manquante

12. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 3 0 1 2
- ☐ 0 0 1 1 2 2 3

13. Le langage C est un langage

- ☐ interprété
- ☐ composé
- ☐ compilé
- ☐ lu, écrit, parlé

14. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#appart <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <studlib.h>`
- ☐ `#include <stdio.h>`

15. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy(p, q);`
- ☐ `n = mccarthy();`
- ☐ `x = mccarthy(n);`

16. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(racine(x)*racine(x));`

17. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 5
- ☐ 3
- ☐ 4

18. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int toto.n = 3;`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `toto_s n, x;`
- ☐ `struct toto_s toto;`

19. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui ne sera pas détectée

20. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ rien
- ☐ Mineur  
Majeur
- ☐ Majeur

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.

- ☐ #define N = 3  
☐ #define N 3  
☐ #define taille = N  
☐ #define taille = 3

2. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1  
☐ 0 1 2 3 4  
☐ 4 3 2 1 0  
☐ 1 2 3 4

3. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop  
☐ un point-virgule manquant  
☐ une accolade en trop  
☐ une accolade manquante

4. Un bit est :

- ☐ la longueur d'un mot mémoire  
☐ un battement d'horloge processeur  
☐ l'instruction qui met fin à un programme  
☐ un chiffre binaire (0 ou 1)

5. Pour déclarer une fonction **saisie\_utilisateur** qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ int saisie\_utilisateur();  
☐ void saisie\_utilisateur(int n);  
☐ void saisie\_utilisateur(char c);  
☐ saisie\_utilisateur(scanf(%d));

6. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(3,5)=8, a=3, b=5  
☐ f(a,b)=13, a=8, b=5  
☐ f(a,b)=8, a=8, b=5  
☐ f(a,b)=8, a=3, b=5

7. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 0  
☐ 5  
☐ 7

8. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
```

```
{
    ...
}
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = %d  
☐ j = 5  
☐ j = 4  
☐ j = 0

9. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur #include manquante  
☐ une fonction déclarée mais non définie  
☐ une fonction appelée avant sa déclaration  
☐ un désaccord entre la déclaration et la définition d'une fonction

10. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a = b  
☐ a == b  
☐ a{n, x} == b{n, x}  
☐ (a.n == b.n) && (a.x == b.x)

11. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8  
☐ 7  
☐ 111  
☐ 3

12. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int toto[5];`
- ☐ `int[] nouveau(5);`
- ☐ `char tableau[5];`
- ☐ `int toto[taille=5];`
- ☐ `int tab[] = 5;`

13. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\* \*\*
- ☐ \*\* \*\* \*\* \*\*
- ☐ \*\*\*\* \*\*
- ☐ \*\* \*\* \*\*

14. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, int x);`

15. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique

16. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
    printf("\n");
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2
- ☐ 0 1 2 0 1 2
- ☐ 1 2 3 1 2
- ☐ 0 1 0 1 0 1

17. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char "c";`

18. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`

- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`

19. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Mineur
- ☐ Mineur Majeur
- ☐ Majeur

20. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6
- ☐ 0 2 4 6 8
- ☐ 0 2 4 6
- ☐ 0 1 2 3 4 5 6 7

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5

2. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 8
- ☐ 0
- ☐ 4

3. Sous unix (ou linux), la commande cd permet de :

- ☐ changer de répertoire courant
- ☐ récupérer un programme arrêté avec la commande ab

- ☐ détruire un fichier
- ☐ jouer de la musique
- ☐ ouvrir un bureau partagé (common desktop)

4. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 5
- ☐ 1
- ☐ 0
- ☐ 4

5. Un bit est :

- ☐ un battement d'horloge processeur
- ☐ un chiffre binaire (0 ou 1)
- ☐ l'instruction qui met fin à un programme
- ☐ la longueur d'un mot mémoire

6. Sous unix (ou linux), la commande ls permet de :

- ☐ compiler un programme
- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ voir des clips musicaux
- ☐ afficher le contenu d'un fichier texte

7. Si n est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ printf("Valeur de n ? %g\n", n);
- ☐ un débogueur
- ☐ printf("Valeur de n ? %d\n", n);
- ☐ scanf("%d", &n);

8. Pour déclarer une fonction factorielle qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ struct int factorielle(int n);
- ☐ int factorielle();
- ☐ int factorielle(int x);
- ☐ int factorielle(double n);

9. Le langage C est un langage

- ☐ interprété
- ☐ composé
- ☐ compilé
- ☐ lu, écrit, parlé

10. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char 'c';
- ☐ int char;
- ☐ char c;
- ☐ char "c";

11. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses blocs
- ☐ ses champs
- ☐ ses cases

12. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ mkdir TP4
- ☐ kwrite TP4
- ☐ new TP4
- ☐ yppasswd

13. Lorsqu'un programme utilise printf ou scanf il faut qu'il contienne l'instruction préprocesseur :

- ☐ #include <studio.h>
- ☐ #appart <stdlib.h>
- ☐ #include <stdio.h>
- ☐ #include <studlib.h>

14. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :
- ☐ 1
  - ☐ 0
  - ☐ 0.5
  - ☐ 1.5
15. Vous utilisez une boucle `while` quand :
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous n'avez pas déclaré de fonction
16. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou référence indéfinie vers « printf »`
- ☐ l'analyse sémantique
  - ☐ l'analyse harmonique
  - ☐ l'édition de liens
  - ☐ l'analyse des entrées clavier
17. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :
- ☐ analyse lexicale
  - ☐ analyse harmonique
  - ☐ analyse sémantique
  - ☐ analyse syntaxique
18. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :
- ☐ des processus
  - ☐ en temps d'accès
  - ☐ certaines données de la mémoire de travail
  - ☐ les fichiers du disque
19. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :
- ☐ `x = racine(x * x) - racine(x);`
  - ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x - 1 = racine(x);`
  - ☐ `x = racine(2/3);`
20. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
- ☐ `printf("%d", factorielle(n));`
  - ☐ `n = factorielle();`
  - ☐ `n = factorielle(p, q);`
  - ☐ `int factorielle(int 2);`

**Barème : 1 point par réponse juste (unique) ; –0,5 point par réponse fausse. Durée : 20 minutes.**

- Une *segmentation fault* est une erreur qui survient lorsque :
  - ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
  - ☐ la division du programme en zones homogènes échoue
  - ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
  - ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- Pour compiler un programme `prog.c`, on utilise la ligne de commande :
  - ☐ `gcc prog.c -o -Wall prog.exe`
  - ☐ `gcc -Wall prog.exe -o prog.c`
  - ☐ `gcc prog.exe -Wall -o prog.c`
  - ☐ `gcc -Wall prog.c -o prog.exe`
- L'ordonnancement par tourniquet permet :
  - ☐ de doubler la mémoire disponible
  - ☐ de ne pas perdre de temps avec la commutation de contexte
  - ☐ d'afficher des ronds colorés à l'écran
  - ☐ d'entretenir l'illusion que les processus tournent en parallèle
- Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :
  - ☐ analyse harmonique
  - ☐ analyse sémantique
  - ☐ analyse syntaxique
  - ☐ analyse lexicale
- Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.
  - ☐ `#define N = 3`
  - ☐ `#define taille = N`
  - ☐ `#define taille = 3`
  - ☐ `#define N 3`

- Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
  - ☐ `int pgcd(int y, int x);`
  - ☐ `int pgcd(int x, y);`
  - ☐ `int pgcd(int x, int x);`
  - ☐ `void pgcd(int x, int y);`

- Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

- Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ la variable `x` vaut  $-\frac{1}{2}$
- ☐ le programme affiche `x`
- ☐ la variable `x` vaut 16
- ☐ le programme affiche `****`

- On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `a<=n<=b`

- Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou référence indéfinie vers < printf >`

- ☐ l'analyse des entrées clavier
- ☐ l'analyse sémantique
- ☐ l'édition de liens
- ☐ l'analyse harmonique

- Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`

- Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
```

```

    printf("*");
}
printf(" ");
}

```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\* \* \* \* \*
- ☐ \* \* \* \* \*
- ☐ \* \* \* \* \*
- ☐ \* \* \* \* \*

13. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char 'c';
- ☐ char "c";
- ☐ char c;
- ☐ int char;

14. Soit la fonction g définie par :

```

int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}

```

Alors l'expression g(0) prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

15. Soit la fonction f définie par :

```

int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}

```

Alors l'expression f(1) prendra la valeur :

- ☐ 0
- ☐ 4
- ☐ 5
- ☐ 1

16. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ void exposant(double x^n);
- ☐ double exposant(double x, int n);
- ☐ exposant(double x, int n, int r);
- ☐ int exposant(double n, int x);

17. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8
- ☐ 7
- ☐ 111
- ☐ 3

18. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée et définie
- ☐ l'avoir définie
- ☐ l'avoir déclarée
- ☐ avoir défini une constante symbolique de la taille de cette fonction

19. Au début de la fonction **main()** on place le code :

```

char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");

```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ A
- ☐ i
- ☐ cccccc

20. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`
  - ☐ `(n<=a) && (n<=b)`
  - ☐ `a<=n<=b`
  - ☐ `(a<n) || (n>b)`
2. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
- ☐ `for(i=0;i<=5;i=i+1)`
  - ☐ `for(i=1;i<5;i=i+1)`
  - ☐ `for(i=0;i<5;i=i+1)`
  - ☐ `for(i=1;i<=5;i=i+1)`
3. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :
- ☐ `printf("x=%d et y=%d\n",x y);`
  - ☐ `printf("x=%d et y=%d\n,x,y");`
  - ☐ `printf("x=%x et y=%y\n");`
  - ☐ `printf("x=%d et y=%d\n",x,y);`
4. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?
- ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `(!A || B)`
  - ☐ `!(A || B) == (A && !B)`
  - ☐ `A && B`

5. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(int n);`
- ☐ `n = carre(n);`
- ☐ `int carre(2);`
- ☐ `int n = carre();`

6. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Mineur Majeur
- ☐ Majeur
- ☐ rien

7. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il ne compile pas

8. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 7

9. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
- ☐ 0 0 0 1 1 1
- ☐ 1 2 1 2 3

10. Un bit est :

- ☐ un battement d'horloge processeur
- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)
- ☐ la longueur d'un mot mémoire

11. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 1 2 3 4
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0

12. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(double n);`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`

13. Sous unix (ou linux), la commande **ls** permet de :

- ☐ voir des clips musicaux
- ☐ compiler un programme
- ☐ afficher le contenu d'un fichier texte
- ☐ afficher la liste de fichiers contenus dans un répertoire

14. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `yppasswd`
- ☐ `mkdir TP4`
- ☐ `kwrite TP4`
- ☐ `new TP4`

15. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque
- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le main

16. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 0
- ☐ la variable x vaut 5 et la variable y vaut 0

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Majeur
- ☐ rien

- ☐ Mineur
- ☐ Mineur
- ☐ Majeur

18. Au début de la fonction **main()** on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A
- ☐ B
- ☐ C
- ☐ b

19. Au début de la fonction **main()** on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ i
- ☐ cccccc
- ☐ A

20. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses blocs
- ☐ ses chants
- ☐ ses cases

**Barème : 1 point par réponse juste (unique) ; –0,5 point par réponse fausse. Durée : 20 minutes.**

- Sous unix (ou linux), la commande `ls` permet de :
  - ☐ compiler un programme
  - ☐ afficher le contenu d'un fichier texte
  - ☐ voir des clips musicaux
  - ☐ afficher la liste de fichiers contenus dans un répertoire
- Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
  - ☐ dans lequel ces fonctions sont appelées dans le `main`
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ alphabétique
  - ☐ un ordre quelconque
- Quel est le problème d'un programme comportant les lignes suivantes ?
 

```
while (1)
{
    printf("coucou\n");
}
```

  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il ne compile pas
  - ☐ il n'affiche rien
  - ☐ il comporte une boucle infinie
- Soit la fonction `f` définie par :
 

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

 Alors l'expression `f(1)` prendra la valeur :

- ☐ 1
  - ☐ 0
  - ☐ 5
  - ☐ 4
- Pour compiler un programme `prog.c`, on utilise la ligne de commande :
    - ☐ `gcc -Wall prog.exe -o prog.c`
    - ☐ `gcc prog.c -o -Wall prog.exe`
    - ☐ `gcc -Wall prog.c -o prog.exe`
    - ☐ `gcc prog.exe -Wall -o prog.c`
  - Quels calculs peut-on programmer en programmation structurée ?
    - ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
    - ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
    - ☐ certains programmes sont de vrais plats de spaghetti
    - ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
  - Une variable booléenne est une variable :
    - ☐ à laquelle une valeur vient d'être affectée
    - ☐ qui est vraie ou fausse
    - ☐ réelle positive
    - ☐ NaN (not a number, qui n'est pas un nombre)
    - ☐ jamais nulle
  - Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
    - ☐ `exposant(double x, int n, int r);`
    - ☐ `void exposant(double x^n);`
    - ☐ `int exposant(double n, int x);`
    - ☐ `double exposant(double x, int n);`

- Un registre du processeur est :
  - ☐ une unité de calcul spécialisée de l'ordinateur
  - ☐ un composant qui contient la liste des fichiers du système
  - ☐ une gamme de fréquence de fonctionnement du processeur
  - ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- Le code suivant :
 

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

 affichera :
  - ☐ 8 6 4 2 0
  - ☐ 0 2 4 6 8
  - ☐ 8 6 4 2
  - ☐ 8 2
- Pour l'extrait de programme suivant :
 

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

 La valeur de somme affichée est :
  - ☐ 16
  - ☐ 6
  - ☐ 3
  - ☐ 20
- L'ordonnancement par tourniquet permet :
  - ☐ d'afficher des ronds colorés à l'écran
  - ☐ d'entretenir l'illusion que les processus tournent en parallèle
  - ☐ de ne pas perdre de temps avec la commutation de contexte
  - ☐ de doubler la mémoire disponible

13. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

14. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A
- ☐ ABCDEF
- ☐ cccccc
- ☐ i

15. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
```

```
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a{n, x} == b{n, x}`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a = b`
- ☐ `a == b`

16. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`

17. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses cases
- ☐ ses chants
- ☐ ses blocs

18. Le langage C est un langage

- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ composé
- ☐ interprété

19. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ `j = 5`
- ☐ `j = 0`
- ☐ `j = 4`
- ☐ `j = %d`

20. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `main`
- ☐ `include`
- ☐ `init`
- ☐ `begin`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`, que doit-on chercher dans le programme ?
  - ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction appelée avant sa déclaration
- Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou référence indéfinie vers « printf »`
  - ☐ l'analyse des entrées clavier
  - ☐ l'analyse harmonique
  - ☐ l'édition de liens
  - ☐ l'analyse sémantique
- On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :
 

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

 Quelle est la condition `cond` :
  - ☐ `(n<=a) && (n<=b)`
  - ☐ `(a<=n) && (n<=b)`
  - ☐ `(a<n) || (n>b)`
  - ☐ `a<=n<=b`
- Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
  - ☐ `for(i=1;i<5;i=i+1)`
  - ☐ `for(i=0;i<=5;i=i+1)`
  - ☐ `for(i=0;i<5;i=i+1)`
  - ☐ `for(i=1;i<=5;i=i+1)`

- On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?
  - ☐ `A && B`
  - ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `!(A || B) == (A && !B)`
  - ☐ `(!A || B)`
- Dans la commande `gcc`, l'option `-Wall` signifie :
  - ☐ qu'il faut lancer un débogueur
  - ☐ qu'on veut changer aléatoirement de fond d'écran
  - ☐ que l'on veut voir tous les avertissements
  - ☐ qu'il faut indenter le fichier source
- Le code suivant :
 

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

 affichera :
  - ☐ 4 3 2 1
  - ☐ 4 3 2 1 0
  - ☐ 0 1 2 3
  - ☐ 0 1 2 3 4
- Si `a` et `b` sont deux variables de type :
 

```
struct toto_s
{
    int n;
    double x;
};
```

 Alors pour tester l'égalité de `a` et de `b` on utilise la condition :
  - ☐ `a = b`
  - ☐ `(a.n == b.n) && (a.x == b.x)`
  - ☐ `a{n, x} == b{n, x}`
  - ☐ `a == b`

- Une variable booléenne est un variable :
  - ☐ jamais nulle
  - ☐ NaN (not a number, qui n'est pas un nombre)
  - ☐ à laquelle une valeur vient d'être affectée
  - ☐ qui est vraie ou fausse
  - ☐ réelle positive
- Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :
  - ☐ `int afficher_menu();`
  - ☐ `void afficher_menu();`
  - ☐ `int afficher_menu(int char);`
  - ☐ `char afficher_menu(printf("menu"));`
  - ☐ `double afficher_menu();`
- Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
  - ☐ `int exposant(double n, int x);`
  - ☐ `exposant(double x, int n, int r);`
  - ☐ `void exposant(double x^n);`
  - ☐ `double exposant(double x, int n);`
- Sur unix (ou linux), la commande `mkdir` permet de :
  - ☐ créer un fichier texte
  - ☐ changer de répertoire courant
  - ☐ créer un répertoire
  - ☐ ouvrir un fichier texte
- Le type des réels en C est :
  - ☐ `double`
  - ☐ `char`
  - ☐ `int`
  - ☐ `real`
- Le bus système sert à :
  - ☐ Transférer des données et intructions entre processeur et mémoire
  - ☐ Écrire des données sur le disque dur
  - ☐ Arriver à l'heure en cours
  - ☐ transporter les processus du tourniquet au processeur

15. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 7

16. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
```

```
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ le terminal affiche 8
- ☐ le bus explose
- ☐ la case mémoire 8 contiendra 0
- ☐ la case mémoire 8 contiendra 16

17. L'ordonnancement par tourniquet permet :

- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ de doubler la mémoire disponible
- ☐ d'afficher des ronds colorés à l'écran
- ☐ de ne pas perdre de temps avec la commutation de contexte

18. Un registre du processeur est :

- ☐ une unité de calcul spécialisée de l'ordinateur

- ☐ un composant qui contient la liste des fichiers du système
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

19. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.exe -o prog.c`

20. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `int factorielle(int 2);`
- ☐ `n = factorielle();`
- ☐ `printf("%d", factorielle(n));`
- ☐ `n = factorielle(p, q);`

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :
  - ☐ en temps d'accès
  - ☐ certaines données de la mémoire de travail
  - ☐ les fichiers du disque
  - ☐ des processus
2. Une variable booléenne est un variable :
  - ☐ jamais nulle
  - ☐ NaN (not a number, qui n'est pas un nombre)
  - ☐ qui est vraie ou fausse
  - ☐ réelle positive
  - ☐ à laquelle une valeur vient d'être affectée
3. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :
  - ☐ A
  - ☐ cccccc
  - ☐ i
  - ☐ ABCDEF
4. Un bit est :
  - ☐ un battement d'horloge processeur
  - ☐ un chiffre binaire (0 ou 1)
  - ☐ la longueur d'un mot mémoire
  - ☐ l'instruction qui met fin à un programme
5. L'écriture 101 en binaire correspond au nombre naturel :
  - ☐ 3
  - ☐ 101
  - ☐ 4
  - ☐ 5

6. Le type des réels en C est :
  - ☐ `char`
  - ☐ `double`
  - ☐ `int`
  - ☐ `real`
7. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :
  - ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x = racine(x * x) - racine(x);`
  - ☐ `x - 1 = racine(x);`
  - ☐ `x = racine(2/3);`
8. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

  - ☐ la variable `y` vaut 5
  - ☐ la variable `x` vaut 5 et la variable `y` vaut 3
  - ☐ la variable `x` vaut 3
  - ☐ le programme affiche "Faux"
9. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :
  - ☐ `x = mccarthy(n);`
  - ☐ `int mccarthy(int 2);`
  - ☐ `n = mccarthy(p, q);`
  - ☐ `n = mccarthy();`
10. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
  - ☐ `for(i=1; i<=5; i=i+1)`
  - ☐ `for(i=0; i<=5; i=i+1)`
  - ☐ `for(i=1; i<5; i=i+1)`
  - ☐ `for(i=0; i<5; i=i+1)`

11. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :
  - ☐ 0
  - ☐ 1
  - ☐ 0.5
  - ☐ 1.5
12. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction déclarée mais non définie
  - ☐ une fonction appelée avant sa déclaration
13. Sous unix (ou linux), la commande `ls` permet de :
  - ☐ voir des clips musicaux
  - ☐ compiler un programme
  - ☐ afficher la liste de fichiers contenus dans un répertoire
  - ☐ afficher le contenu d'un fichier texte
14. Avant de faire appel à une fonction il est nécessaire de :
  - ☐ l'avoir définie
  - ☐ l'avoir déclarée
  - ☐ l'avoir déclarée et définie
  - ☐ avoir défini une constante symbolique de la taille de cette fonction
15. Le langage C est un langage
  - ☐ compilé
  - ☐ interprété
  - ☐ lu, écrit, parlé
  - ☐ composé

16. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale
- ☐ analyse harmonique
- ☐ analyse sémantique
- ☐ analyse syntaxique

17. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

- ☐ printf("x=%x et y=%y\n");
- ☐ printf("x=%d et y=%d\n",x y);
- ☐ printf("x=%d et y=%d\n,x,y");
- ☐ printf("x=%d et y=%d\n",x,y);

18. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
```

```
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

19. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a{n, x} == b{n, x}
- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a = b
- ☐ a == b

20. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?
  - ☐ une accolade manquante
  - ☐ un point-virgule en trop
  - ☐ une accolade en trop
  - ☐ un point-virgule manquant
- Un enregistrement permet de grouper plusieurs valeurs dans :
  - ☐ ses chants
  - ☐ ses blocs
  - ☐ ses champs
  - ☐ ses cases
- Un registre du processeur est :
  - ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
  - ☐ un composant qui contient la liste des fichiers du système
  - ☐ une gamme de fréquence de fonctionnement du processeur
  - ☐ une unité de calcul spécialisée de l'ordinateur
- Après exécution du programme :
 

```

1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
      
```

  - ☐ le terminal affiche 8
  - ☐ la case mémoire 8 contiendra 0
  - ☐ le bus explose
  - ☐ la case mémoire 8 contiendra 16

- Un bit est :
  - ☐ un battement d'horloge processeur
  - ☐ un chiffre binaire (0 ou 1)
  - ☐ l'instruction qui met fin à un programme
  - ☐ la longueur d'un mot mémoire
- Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :
  - ☐ `x = racine(2/3);`
  - ☐ `x - 1 = racine(x);`
  - ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x = racine(x * x) - racine(x);`
- Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
  - ☐ `exposant(double x, int n, int r);`
  - ☐ `void exposant(double x^n);`
  - ☐ `double exposant(double x, int n);`
  - ☐ `int exposant(double n, int x);`
- Une *segmentation fault* est une erreur qui survient lorsque :
  - ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
  - ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
  - ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
  - ☐ la division du programme en zones homogènes échoue
- On souhaite faire une boucle de contrôle de saisie : tant que l'entier **n** n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de **n**. Soit le programme suivant :
 

```

int a = 0;
int b = 20;
int n;
      
```

```

scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
      
```

Quelle est la condition **cond** :

- ☐ `(n<=a) && (n<=b)`
  - ☐ `(a<=n) && (n<=b)`
  - ☐ `a<=n<=b`
  - ☐ `(a<n) || (n>b)`
- Pour déclarer une procédure **afficher\_date** qui prend en argument un **struct date\_s** et affiche le contenu du struct, on écrit :
    - ☐ `struct date_s afficher_date(struct date_s d);`
    - ☐ `void afficher_date(struct date_s d);`
    - ☐ `void afficher_date(date_s d);`
    - ☐ `int afficher_date(date_s d);`
  - Les lignes
 

```

int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
      
```

    - ☐ comportent une erreur qui ne sera pas détectée
    - ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
    - ☐ ne comportent aucune erreur
    - ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
  - Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :
    - ☐ analyse sémantique
    - ☐ analyse lexicale
    - ☐ analyse harmonique
    - ☐ analyse syntaxique

13. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=8, a=3, b=5

14. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 1
- ☐ 5

15. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

- ☐ la variable y vaut 5
- ☐ la variable x vaut 3
- ☐ la variable x vaut 5 et la variable y vaut 3
- ☐ le programme affiche "Faux"

16. L'ordonnancement par tourniquet permet :

- ☐ de doubler la mémoire disponible
- ☐ d'afficher des ronds colorés à l'écran
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'entretenir l'illusion que les processus tournent en parallèle

17. Le langage C est un langage

- ☐ compilé
- ☐ composé
- ☐ lu, écrit, parlé
- ☐ interprété

18. Si x est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 0
- ☐ 0.5
- ☐ 1.5
- ☐ 1

19. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur **#include** manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie

20. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 0
- ☐ 3
- ☐ 4

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut indenter le fichier source
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut lancer un débogueur

2. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 0 0 1 1 1
- ☐ 1 2 1 2 3

3. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 0 1 2 3
- ☐ 0 1 2 3 4

4. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses chants
- ☐ ses champs
- ☐ ses blocs

5. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration

6. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 0
- ☐ 16
- ☐ 4

7. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `x = mccarthy(n);`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy();`
- ☐ `n = mccarthy(p, q);`

8. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ la division du programme en zones homogènes échoue

9. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(date_s d);`

10. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 0
- ☐ 5

11. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l’affichage sera :

- ☐ ABCDEF
- ☐ i
- ☐ cccccc
- ☐ A

12. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L’ordre dans lequel vous devez maintenant définir ces fonctions est l’ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique

13. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu’est ce qui sera affiché ?

- ☐ j = 5
- ☐ j = 0
- ☐ j = %d
- ☐ j = 4

14. Le langage C est un langage

- ☐ interprété
- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ composé

15. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`
- ☐ `double exposant(double x, int n);`
- ☐ `void exposant(double x^n);`
- ☐ `int exposant(double n, int x);`

16. Afin de représenter la taille d’un tableau, définir une constante symbolique N valant 3.

- ☐ `#define N = 3`
- ☐ `#define taille = N`
- ☐ `#define taille = 3`
- ☐ `#define N 3`

17. Laquelle de ces écritures correspond à la déclaration d’une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`

18. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`
- ☐ `int factorielle(double n);`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle();`

19. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Mineur  
Majeur
- ☐ rien

20. Après exécution jusqu’à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ le programme affiche "Faux"
- ☐ la variable y vaut 5
- ☐ la variable x vaut 0
- ☐ la variable x vaut 5 et la variable y vaut 0

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 1 2 3 4
- ☐ 0 1 2 3 4

2. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses chants
- ☐ ses cases
- ☐ ses blocs

3. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 5; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = %d
- ☐ j = 5
- ☐ j = 0
- ☐ j = 4

4. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `int saisie_utilisateur();`
- ☐ `saisie_utilisateur(scanf(%d));`

5. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

6. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale
- ☐ analyse harmonique
- ☐ analyse sémantique
- ☐ analyse syntaxique

7. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque

- ☐ dans lequel ces fonctions sont appelées dans le `main`

- ☐ alphabétique

8. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu(int char);`
- ☐ `void afficher_menu();`
- ☐ `double afficher_menu();`
- ☐ `int afficher_menu();`
- ☐ `char afficher_menu(printf("menu"));`

9. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours
- ☐ transporter les processus du tourniquet au processeur

10. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche `****`
- ☐ la variable `x` vaut  $-\frac{1}{2}$
- ☐ la variable `x` vaut 16
- ☐ le programme affiche `x`

11. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal

12. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
- ☐ `#include <studio.h>`
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <studlib.h>`
  - ☐ `#include <stdio.h>`
13. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?
- ☐ `(!A || B)`
  - ☐ `!(!A || B) == (A && !B)`
  - ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `A && B`
14. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
- ☐ `#include <studlib.h>`
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <stdio.h>`
  - ☐ `#include <studio.h>`
15. Les lignes
- ```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```
- ☐ comportent une erreur qui ne sera pas détectée

- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
  - ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
  - ☐ ne comportent aucune erreur
16. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
- ☐ `int factorielle(int 2);`
  - ☐ `n = factorielle(p, q);`
  - ☐ `printf("%d", factorielle(n));`
  - ☐ `n = factorielle();`
17. Une variable booléenne est un variable :
- ☐ réelle positive
  - ☐ NaN (not a number, qui n'est pas un nombre)
  - ☐ jamais nulle
  - ☐ à laquelle une valeur vient d'être affectée
  - ☐ qui est vraie ou fausse
18. Pour l'extrait de programme suivant :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```
- qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
  - ☐ 0 0 1 1 2 2 3
  - ☐ 0 1 2 0 1 2 3
  - ☐ 0 1 2 3 0 1 2
19. Soit la fonction `f` définie par :
- ```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```
- Alors l'expression `f(0)` prendra la valeur :
- ☐ 4
  - ☐ 3
  - ☐ 0
20. Un fichier source est :
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
  - ☐ un document qui doit être protégé
  - ☐ un document illisible pour les humains
  - ☐ un document de référence du système
  - ☐ un fichier texte qui sera traduit en instructions processeur

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0

2. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7
- ☐ 3
- ☐ 8
- ☐ 111

3. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3
- ☐ 16
- ☐ 6
- ☐ 20

4. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `double exposant(double x, int n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`
- ☐ `void exposant(double x^n);`

5. Le langage C est un langage

- ☐ compilé
- ☐ composé
- ☐ lu, écrit, parlé
- ☐ interprété

6. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction **f** ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

7. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`

8. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `scanf("%d", &n);`
- ☐ un débogueur

9. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration

10. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut lancer un débogueur
- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran

11. Pour déclarer une fonction **pgcd** qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int x, int x);`

12. Si **carre** est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que  $n$  est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`
- ☐ `int carre(2);`
- ☐ `int n = carre();`

13. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée et définie
- ☐ l'avoir définie
- ☐ l'avoir déclarée
- ☐ avoir défini une constante symbolique de la taille de cette fonction

14. Vous utilisez une boucle **while** quand :

- ☐ vous avez déjà fait un **for** dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

15. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 0
- ☐ 16
- ☐ 8
- ☐ 4

16. Sous unix (ou linux), la commande **ls** permet de :

- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ compiler un programme
- ☐ afficher le contenu d'un fichier texte
- ☐ voir des clips musicaux

17. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse lexicale
- ☐ analyse sémantique
- ☐ analyse syntaxique

18. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ **Majeur**
- ☐ **Mineur**  
**Majeur**
- ☐ **Mineur**

19. Une variable booléenne est un variable :

- ☐ réelle positive
- ☐ à laquelle une valeur vient d'être affectée
- ☐ jamais nulle
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ qui est vraie ou fausse

20. Un registre du processeur est :

- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une unité de calcul spécialisée de l'ordinateur



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main

2. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

3. Sous unix (ou linux), la commande `cd` permet de :

- ☐ changer de répertoire courant
- ☐ détruire un fichier
- ☐ jouer de la musique
- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ ouvrir un bureau partagé (common desktop)

4. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `int char;`

5. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\* \* \* \* \*
- ☐ \*\* \*\* \*\* \*\* \*\* \*\* \*\*
- ☐ \*\*\*\*\* \*\* \*
- ☐ \*\* \*\*\* \*\* \*

6. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(2/3);`

7. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses champs
- ☐ ses cases
- ☐ ses chants

8. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`

9. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 0
- ☐ 4
- ☐ 8

10. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie

11. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `int factorielle(int 2);`
- ☐ `n = factorielle(p, q);`
- ☐ `n = factorielle();`
- ☐ `printf("%d", factorielle(n));`

12. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 3 0 1 2
- ☐ 0 1 2 0 1 2

13. Un fichier source est :

- ☐ un document de référence du système
- ☐ un document illisible pour les humains
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document qui doit être protégé

14. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 0
- ☐ la case mémoire 8 contiendra 16
- ☐ le terminal affiche 8
- ☐ le bus explose

15. Si cette erreur apparaît à la compilation :

**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop
- ☐ une accolade manquante
- ☐ un point-virgule manquant
- ☐ une accolade en trop

16. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <studio.h>`
- ☐ `#include <studlib.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <stdio.h>`

17. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `double exposant(double x, int n);`
- ☐ `int exposant(double n, int x);`

18. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

19. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu(int char);`
- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu();`
- ☐ `void afficher_menu();`
- ☐ `double afficher_menu();`

20. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

☐ `int k;`  
☐ `loop i;`  
☐ `int loop n;`  
☐ `int %d;`

2. Le type des réels en C est :

☐ `double`  
☐ `char`  
☐ `real`  
☐ `int`

3. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

☐ Mineur  
☐ Majeur  
☐ rien  
☐ Mineur  
☐ Majeur

4. Vous utilisez une boucle `while` quand :

☐ vous avez déjà fait un `for` dans le même programme principal  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ vous n'avez pas déclaré de fonction  
☐ l'incrément de la variable de boucle n'est pas 1

5. Le langage C est un langage

☐ compilé  
☐ interprété  
☐ lu, écrit, parlé  
☐ composé

6. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

☐ il ne compile pas  
☐ il comporte une boucle infinie  
☐ il n'affiche rien  
☐ il risque d'afficher bonjour à la place de coucou

7. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

☐ 4 3 2 1 0  
☐ 0 1 2 3 4  
☐ 1 2 3 4  
☐ 4 3 2 1

8. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

☐ 0 1 2 0 1 2 3  
☐ 0 0 1 1 2 2 3  
☐ 0 1 2 3 0 1 2  
☐ 0 1 2 0 1 2

9. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

☐ 0.5  
☐ 0  
☐ 1.5  
☐ 1

10. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n", f(a,b), a, b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=8, a=3, b=5`  
☐ `f(a,b)=13, a=8, b=5`  
☐ `f(3,5)=8, a=3, b=5`

11. Un registre du processeur est :

☐ une gamme de fréquence de fonctionnement du processeur  
☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs  
☐ une unité de calcul spécialisée de l'ordinateur  
☐ un composant qui contient la liste des fichiers du système

12. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ dans lequel vous avez déclaré ces fonction
  - ☐ alphabétique
  - ☐ un ordre quelconque
  - ☐ dans lequel ces fonctions sont appelées dans le main
13. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max', que doit-on chercher dans le programme?`
- ☐ une fonction appelée avant sa déclaration
  - ☐ une fonction déclarée mais non définie
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
14. Pour compiler un programme `prog.c`, on utilise la ligne de commande :
- ☐ `gcc prog.c -o -Wall prog.exe`
  - ☐ `gcc -Wall prog.c -o prog.exe`
  - ☐ `gcc prog.exe -Wall -o prog.c`
  - ☐ `gcc -Wall prog.exe -o prog.c`
15. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :
- ```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`
  - ☐ `(n<=a) && (n<=b)`
  - ☐ `a<=n<=b`
  - ☐ `(a<n) || (n>b)`
16. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
- ☐ `scanf("%d", &n);`
  - ☐ `printf("Valeur de n ? %g\n", n);`
  - ☐ un débogueur
  - ☐ `printf("Valeur de n ? %d\n", n);`
17. Pour l'extrait de programme suivant :
- ```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```
- La valeur de somme affichée est :
- ☐ 6
  - ☐ 3
  - ☐ 20
  - ☐ 16
18. Soit la fonction `f` définie par :
- ```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1
  - ☐ 0
  - ☐ 4
  - ☐ 5
19. Le code suivant :
- ```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 0 1 2 3
  - ☐ 4 3 2 1
  - ☐ 0 1 2 3 4
  - ☐ 4 3 2 1 0
20. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```
- ☐ la variable `x` vaut  $-\frac{1}{2}$
  - ☐ le programme affiche `x`
  - ☐ la variable `x` vaut 16
  - ☐ le programme affiche `****`

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 1 2 3 4 5 6
- ☐ 0 2 4 6

2. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y = 3;
13
14     x = y;
15
16     ...
17 }
```

- ☐ la variable x vaut 5 et la variable y vaut 3
- ☐ la variable x vaut 3
- ☐ le programme affiche "Faux"
- ☐ la variable y vaut 5

3. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui ne sera pas détectée

4. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 1
- ☐ 5
- ☐ 4
- ☐ 0

5. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 4
- ☐ 5
- ☐ 3

6. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ x = racine(racine(x)\*racine(x));
- ☐ x = racine(x \* x) - racine(x);
- ☐ x = racine(2/3);
- ☐ x - 1 = racine(x);

7. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses blocs
- ☐ ses cases
- ☐ ses chants

8. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop
- ☐ une accolade manquante
- ☐ un point-virgule manquant
- ☐ une accolade en trop

9. Une variable booléenne est un variable :

- ☐ jamais nulle
- ☐ à laquelle une valeur vient d'être affectée
- ☐ qui est vraie ou fausse
- ☐ réelle positive
- ☐ NaN (not a number, qui n'est pas un nombre)

10. Le bus système sert à :

- ☐ Arriver à l'heure en cours
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur

11. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 0
- ☐ 1
- ☐ 42
- ☐ 6

12. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%x et y=%y\n");`

13. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(struct date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`

14. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ ouvrir un fichier texte
- ☐ créer un répertoire
- ☐ créer un fichier texte
- ☐ changer de répertoire courant

15. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle();`
- ☐ `n = factorielle(p, q);`
- ☐ `printf("%d", factorielle(n));`
- ☐ `int factorielle(int 2);`

16. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ rien

17. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 3

18. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`
- ☐ `int n = carre();`
- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`

19. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ alphabétique
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction

20. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle();`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle(double n);`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le langage C est un langage

- ☐ interprété
- ☐ lu, écrit, parlé
- ☐ compilé
- ☐ composé

2. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.

- ☐ `#define N = 3`
- ☐ `#define taille = 3`
- ☐ `#define taille = N`
- ☐ `#define N 3`

3. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1
- ☐ 0 0 1 1 2 2
- ☐ 0 1 2 0 1 2
- ☐ 1 2 3 1 2

4. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `char c;`
- ☐ `char "c";`
- ☐ `int char;`

5. Après exécution jusqu'à la ligne 15 du programme C :

```
10  ...
11  int main() {
12      int x = 5;
13
14      x = 3 * x + 1;
15
16      ...
17  }
```

- ☐ la variable x vaut 16
- ☐ le programme affiche x
- ☐ le programme affiche \*\*\*\*
- ☐ la variable x vaut  $-\frac{1}{2}$

6. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Mineur Majeur
- ☐ rien

7. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 0
- ☐ 6
- ☐ 42
- ☐ 1

8. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche x = 2
- ☐ le terminal affiche x = 5
- ☐ le terminal affiche 5
- ☐ le terminal affiche "Faux"

9. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`
- ☐ `void exposant(double x^n);`
- ☐ `double exposant(double x, int n);`
- ☐ `int exposant(double n, int x);`

10. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`
- ☐ `a == b`

11. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 5
- ☐ j = %d
- ☐ j = 4
- ☐ j = 0

12. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours
- ☐ transporter les processus du tourniquet au processeur
- ☐ Écrire des données sur le disque dur

13. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
}
```

```
return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

14. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 111
- ☐ 8
- ☐ 7

15. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur **#include** manquante

16. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque
- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le main

17. Pour déclarer une fonction **saisie\_utilisateur** qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ **int** saisie\_utilisateur();
- ☐ **void** saisie\_utilisateur(char c);
- ☐ **void** saisie\_utilisateur(int n);
- ☐ **saisie\_utilisateur**(scanf(%d));

18. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ init
- ☐ main
- ☐ include
- ☐ begin

19. Au début de la fonction **main()** on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ cccccc
- ☐ ABCDEF
- ☐ i
- ☐ A

20. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ **struct** int factorielle(int n);
- ☐ **int** factorielle(double n);
- ☐ **int** factorielle();
- ☐ **int** factorielle(int x);



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur  
☐ Majeur  
☐ Mineur  
☐ Majeur  
☐ rien

2. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`  
☐ `#appart <stdlib.h>`  
☐ `#include <stdio.h>`  
☐ `#include <studio.h>`

3. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `kwrite TP4`  
☐ `mkdir TP4`  
☐ `new TP4`  
☐ `yppasswd`

4. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ `(A == TRUE) && (B == TRUE)`  
☐ `!(A || B) == (A && !B)`  
☐ `A && B`  
☐ `(!A || B)`

5. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42  
☐ 1  
☐ 0  
☐ 6

6. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ B  
☐ C  
☐ A  
☐ b

7. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche 5  
☐ le terminal affiche "Faux"  
☐ le terminal affiche x = 5  
☐ le terminal affiche x = 2

8. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4  
☐ 0  
☐ 3

9. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases  
☐ ses blocs  
☐ ses champs  
☐ ses chants

10. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `int afficher_date(date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`  
☐ `void afficher_date(struct date_s d);`  
☐ `void afficher_date(date_s d);`

11. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut lancer un débogueur  
☐ qu'il faut indenter le fichier source  
☐ que l'on veut voir tous les avertissements  
☐ qu'on veut changer aléatoirement de fond d'écran

12. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 16
- ☐ 0
- ☐ 4

13. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ un débogueur
- ☐ `printf("Valeur de n ? %d\n", n);`

14. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `double afficher_menu();`
- ☐ `char afficher_menu printf("menu");`
- ☐ `int afficher_menu();`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu(int char);`

15. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `int char;`
- ☐ `char c;`
- ☐ `char "c";`
- ☐ `char 'c';`

16. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy(p, q);`
- ☐ `n = mccarthy();`
- ☐ `int mccarthy(int 2);`

17. Si cette erreur apparaît à la compilation : `error: expected ';' before '}' token` que doit-on chercher dans le programme?

- ☐ un point-virgule manquant
- ☐ une accolade en trop
- ☐ une accolade manquante
- ☐ un point-virgule en trop

18. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

19. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#appart <stdlib.h>`
- ☐ `#include <stdio.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <studlib.h>`

20. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 16
- ☐ 3
- ☐ 20
- ☐ 6

**Barème : 1 point par réponse juste (unique) ; -0,5 point par réponse fausse. Durée : 20 minutes.**

1. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`  
☐ `n = carre(int n);`  
☐ `n = carre(n);`  
☐ `int n = carre();`

2. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ ne comportent aucune erreur  
☐ comportent une erreur qui ne sera pas détectée  
☐ comportent une erreur qui sera détectée au cours de l'édition de lien  
☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique

3. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4  
☐ 5  
☐ 1  
☐ 0

4. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant  
☐ une accolade en trop  
☐ un point-virgule en trop  
☐ une accolade manquante

5. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie  
☐ une directive préprocesseur `#include` manquante  
☐ une fonction appelée avant sa déclaration  
☐ un désaccord entre la déclaration et la définition d'une fonction

6. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien  
☐ il risque d'afficher bonjour à la place de coucou  
☐ il comporte une boucle infinie  
☐ il ne compile pas

7. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche `x = 5`  
☐ le terminal affiche 5  
☐ le terminal affiche `x = 2`  
☐ le terminal affiche "Faux"

8. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `int afficher_date(date_s d);`  
☐ `void afficher_date(struct date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`  
☐ `void afficher_date(date_s d);`

9. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16  
☐ 0  
☐ 4  
☐ 8

10. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur  
☐ Arriver à l'heure en cours  
☐ Transférer des données et intructions entre processeur et mémoire  
☐ Écrire des données sur le disque dur

11. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `int factorielle(int 2);`  
☐ `n = factorielle();`  
☐ `printf("%d", factorielle(n));`  
☐ `n = factorielle(p, q);`

12. Une variable booléenne est un variable :

- ☐ réelle positive  
☐ jamais nulle  
☐ NaN (not a number, qui n'est pas un nombre)  
☐ à laquelle une valeur vient d'être affectée  
☐ qui est vraie ou fausse

13. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char "c";`

14. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ changer de répertoire courant
- ☐ ouvrir un fichier texte
- ☐ créer un fichier texte
- ☐ créer un répertoire

15. Au début de la fonction `main()` on place le code :

```
char b = 'A';  
b = b + 2;  
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b
- ☐ B
- ☐ A
- ☐ C

16. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...  
11    int main() {  
12        int x = 5;  
13  
14        x = 3 * x + 1;  
15  
16        ...  
17    }
```

- ☐ le programme affiche x
- ☐ la variable x vaut  $-\frac{1}{2}$
- ☐ le programme affiche \*\*\*\*
- ☐ la variable x vaut 16

17. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses champs
- ☐ ses chants
- ☐ ses cases

18. Un bit est :

- ☐ un battement d'horloge processeur

- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)

19. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ `(!A || B)`
- ☐ `A && B`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `!(!A || B) == (A && !B)`

20. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ mettre les blocs en séquence les uns à la suite des autres
- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ répéter un bloc tant qu'une condition est vérifiée
- ☐ retourner un bloc

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop
- ☐ une accolade en trop
- ☐ un point-virgule manquant
- ☐ une accolade manquante

2. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ retourner un bloc
- ☐ répéter un bloc tant qu'une condition est vérifiée
- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ mettre les blocs en séquence les uns à la suite des autres

3. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

4. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur

5. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0

6. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Mineur Majeur
- ☐ rien

7. Lorsqu'un programme utilise **printf** ou **scanf** il faut qu'il contienne l'instruction préprocesseur :

- ☐ **#include <studio.h>**
- ☐ **#include <stdio.h>**
- ☐ **#appart <stdlib.h>**
- ☐ **#include <studlib.h>**

8. Si x est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1
- ☐ 0.5
- ☐ 1.5
- ☐ 0

9. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ **(a.n == b.n) && (a.x == b.x)**
- ☐ **a = b**
- ☐ **a == b**
- ☐ **a{n, x} == b{n, x}**

10. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ **char "c";**
- ☐ **char 'c';**
- ☐ **char c;**
- ☐ **int char;**

11. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 5
- ☐ 4
- ☐ 101

12. Vous utilisez une boucle **while** quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un **for** dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

13. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 20
- ☐ 6
- ☐ 3
- ☐ 16

14. Si cet avertissement apparaît à la compilation :

**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur **#include** manquante
- ☐ une fonction déclarée mais non définie

15. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `toto_s n, x;`

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `int toto.n = 3;`
- ☐ `struct toto_s toto;`

16. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ le programme affiche "Faux"
- ☐ la variable y vaut 5
- ☐ la variable x vaut 5 et la variable y vaut 0
- ☐ la variable x vaut 0

17. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

18. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6 8
- ☐ 0 2 4 6
- ☐ 0 1 2 3 4 5 6

19. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.

- ☐ `#define N = 3`
- ☐ `#define taille = 3`
- ☐ `#define N 3`
- ☐ `#define taille = N`

20. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`
- ☐ `int pgcd(2);`
- ☐ `n = pgcd(n, 3);`
- ☐ `n = pgcd(int p, int q);`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Avant de faire appel à une fonction il est nécessaire de :

- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir définie
- ☐ l'avoir déclarée et définie
- ☐ l'avoir déclarée

2. L'ordonnancement par tourniquet permet :

- ☐ d'afficher des ronds colorés à l'écran
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ de doubler la mémoire disponible
- ☐ d'entretenir l'illusion que les processus tournent en parallèle

3. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4

4. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(x * x) - racine(x);`

5. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`
- ☐ `int pgcd(2);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `int n = pgcd();`

6. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »**

- ☐ l'analyse harmonique
- ☐ l'édition de liens
- ☐ l'analyse des entrées clavier
- ☐ l'analyse sémantique

7. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

8. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s n, x;`
- ☐ `int toto.n = 3;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `struct toto_s toto;`

9. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <stdio.h>`

10. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ certaines données de la mémoire de travail
- ☐ en temps d'accès
- ☐ des processus
- ☐ les fichiers du disque

11. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(date_s d);`

12. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 111
- ☐ 3
- ☐ 8
- ☐ 7

13. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse sémantique
- ☐ analyse lexicale
- ☐ analyse syntaxique

14. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu();`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `double afficher_menu();`

15. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.c -o prog.exe`

16. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 5
- ☐ 0

17. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `int char;`

18. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur
- ☐ Écrire des données sur le disque dur
- ☐ Arriver à l'heure en cours
- ☐ Transférer des données et intructions entre processeur et mémoire

19. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <studio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studlib.h>`
- ☐ `#include <stdio.h>`

20. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce `printf` ?

- ☐ `j = 4`
- ☐ `j = 5`
- ☐ `j = %d`
- ☐ `j = 0`



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x - 1 = racine(x);`  
☐ `x = racine(2/3);`  
☐ `x = racine(racine(x)*racine(x));`  
☐ `x = racine(x * x) - racine(x);`

2. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou  
☐ il n'affiche rien  
☐ il comporte une boucle infinie  
☐ il ne compile pas

3. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ B  
☐ C  
☐ A  
☐ b

4. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define N = 3`  
☐ `#define taille = 3`  
☐ `#define N 3`  
☐ `#define taille = N`

5. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce `printf` ?

- ☐ j = 5  
☐ j = %d  
☐ j = 0  
☐ j = 4

6. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le `main`  
☐ un ordre quelconque  
☐ alphabétique  
☐ dans lequel vous avez déclaré ces fonction

7. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `char afficher_menu(printf("menu"));`  
☐ `void afficher_menu();`  
☐ `double afficher_menu();`  
☐ `int afficher_menu();`  
☐ `int afficher_menu(int char);`

8. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a = b`  
☐ `(a.n == b.n) && (a.x == b.x)`  
☐ `a{n, x} == b{n, x}`  
☐ `a == b`

9. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8  
☐ 8 6 4 2 0  
☐ 8 6 4 2  
☐ 8 2

10. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `struct toto_s toto;`  
☐ `toto_s n, x;`  
☐ `int toto.n = 3;`  
☐ `int struct toto_s = {3, -1e10};`  
☐ `toto_s struct z = {3, 0.5};`

11. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
```

```

    return 3;
}
return 4;
}

```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 3
- ☐ 4

12. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse syntaxique
- ☐ analyse lexicale
- ☐ analyse sémantique

13. Soit la fonction `g` définie par :

```

int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}

```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 0
- ☐ 5

14. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ la division du programme en zones homogènes échoue
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée

15. Après exécution jusqu'à la ligne 15 du programme C :

```

10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }

```

- ☐ le programme affiche `x`
- ☐ la variable `x` vaut 16
- ☐ le programme affiche `****`
- ☐ la variable `x` vaut  $-\frac{1}{2}$

16. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(struct date_s d);`

17. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un `for` dans le même programme principal

18. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(!A || B)`
- ☐ `A && B`
- ☐ `!(A || B) == (A && !B)`
- ☐ `(A == TRUE) && (B == TRUE)`

19. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `include`
- ☐ `init`
- ☐ `begin`
- ☐ `main`

20. Quel est l'opérateur de différence en C :

- ☐ `<=`
- ☐ `!`
- ☐ `!=`
- ☐ `<>`

**Barème : 1 points par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char c;  
☐ char "c";  
☐ int char;  
☐ char 'c';

2. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ kwrite TP4  
☐ new TP4  
☐ mkdir TP4  
☐ yppasswd

3. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs  
☐ ses chants  
☐ ses cases  
☐ ses champs

4. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = 5  
☐ j = 0  
☐ j = 4  
☐ j = %d

5. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 7  
☐ 5  
☐ 0

6. Si factorielle est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ printf("%d", factorielle(n));  
☐ n = factorielle(p, q);  
☐ n = factorielle();  
☐ int factorielle(int 2);

7. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 4  
☐ 0  
☐ 3

8. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ double exposant(double x, int n);  
☐ exposant(double x, int n, int r);  
☐ int exposant(double n, int x);  
☐ void exposant(double x^n);

9. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2 0  
☐ 8 6 4 2  
☐ 8 2  
☐ 0 2 4 6 8

10. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2  
☐ 0 1 0 1 0 1 0 1  
☐ 1 2 1 2 3  
☐ 0 0 0 1 1 1

11. Sur unix (ou linux), la commande **mkdir** permet de :

- ☐ changer de répertoire courant  
☐ ouvrir un fichier texte  
☐ créer un répertoire  
☐ créer un fichier texte

12. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4

13. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`

14. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 0
- ☐ 6
- ☐ 1
- ☐ 42

15. Sous unix (ou linux), la commande `cd` permet de :

- ☐ jouer de la musique
- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ détruire un fichier
- ☐ changer de répertoire courant
- ☐ ouvrir un bureau partagé (common desktop)

16. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

17. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 0
- ☐ 1
- ☐ 0.5
- ☐ 1.5

18. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy();`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy(p, q);`

19. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`

20. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

- ☐ le programme affiche "Faux"
- ☐ la variable `x` vaut 5 et la variable `y` vaut 3
- ☐ la variable `x` vaut 3
- ☐ la variable `y` vaut 5

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un fichier source est :

- ☐ un document qui doit être protégé
- ☐ un document de référence du système
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document illisible pour les humains
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur

2. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A
- ☐ b
- ☐ C
- ☐ B

3. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

4. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ certaines données de la mémoire de travail
- ☐ les fichiers du disque
- ☐ des processus
- ☐ en temps d'accès

5. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <studio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <stdio.h>`
- ☐ `#include <studlib.h>`

6. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 0 1 2 3
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1

7. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine

8. On considère deux variables booléennes A et B initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `!(A || B) == (A && !B)`
- ☐ `A && B`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `(!A || B)`

9. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7
- ☐ 111
- ☐ 8
- ☐ 3

10. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy(p, q);`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy();`

11. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 0 0 1 1 1
- ☐ 0 1 2 0 1 2
- ☐ 1 2 1 2 3

12. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu(int char);`
- ☐ `double afficher_menu();`
- ☐ `int afficher_menu();`
- ☐ `void afficher_menu();`

13. Si cette erreur apparaît à la compilation :  
`Undefined symbols : "_printf" ou référence indéfinie vers « printf »` que doit-on chercher dans le programme ?
- ☐ une variable non déclarée
  - ☐ un caractère interdit en C
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une faute de frappe dans un appel de fonction
14. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :
- ☐ analyse syntaxique
  - ☐ analyse sémantique
  - ☐ analyse harmonique
  - ☐ analyse lexicale
15. Dans la commande gcc, l'option `-Wall` signifie :
- ☐ qu'il faut lancer un débogueur
  - ☐ que l'on veut voir tous les avertissements
  - ☐ qu'il faut indenter le fichier source
  - ☐ qu'on veut changer aléatoirement de fond d'écran

16. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ dans lequel ces fonctions sont appelées dans le `main`
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ un ordre quelconque
  - ☐ alphabétique
17. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
- ☐ `char c;`
  - ☐ `char 'c';`
  - ☐ `int char;`
  - ☐ `char "c";`
18. Le bus système sert à :
- ☐ Transférer des données et intructions entre processeur et mémoire
  - ☐ transporter les processus du tourniquet au processeur
  - ☐ Arriver à l'heure en cours
  - ☐ Écrire des données sur le disque dur

19. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 3
- ☐ 4

20. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1

**Barème : 1 points par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si cette erreur apparaît à la compilation :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ une faute de frappe dans un appel de fonction
- ☐ un caractère interdit en C
- ☐ une variable non déclarée

2. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 3
- ☐ 20
- ☐ 16

3. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, int x);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, y);`

4. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ en temps d'accès
- ☐ des processus
- ☐ certaines données de la mémoire de travail
- ☐ les fichiers du disque

5. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1
- ☐ 0 1 2 3

6. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n",x,y);`

7. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut lancer un débogueur

8. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`

9. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 16
- ☐ le bus explose
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0

10. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un `for` dans le même programme principal

11. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée

12. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

13. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée
- ☐ l'avoir définie
- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir déclarée et définie

14. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `struct toto_s toto;`

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `int toto.n = 3;`
- ☐ `toto_s n, x;`

15. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`
- ☐ `n = carre(int n);`
- ☐ `n = carre(n);`
- ☐ `int n = carre();`

16. Un bit est :

- ☐ un battement d'horloge processeur
- ☐ un chiffre binaire (0 ou 1)
- ☐ l'instruction qui met fin à un programme
- ☐ la longueur d'un mot mémoire

17. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
- ☐ 0 0 1 1 2 2
- ☐ 1 2 3 1 2

18. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `scanf("%d", &n);`
- ☐ un débogueur

19. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 4
- ☐ 1

20. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy();`
- ☐ `n = mccarthy(p, q);`
- ☐ `x = mccarthy(n);`



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 4  
☐ j = 0  
☐ j = %d  
☐ j = 5

2. Quels calculs peut-on programmer en programmation structurée ?

- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine  
☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine  
☐ certains programmes sont de vrais plats de spaghetti  
☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée

3. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ mkdir TP4  
☐ kwrite TP4  
☐ new TP4  
☐ yppasswd

4. Vous utilisez une boucle while quand :

- ☐ vous n'avez pas déclaré de fonction  
☐ vous avez déjà fait un for dans le même programme principal  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ l'incrément de la variable de boucle n'est pas 1

5. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 0  
☐ 5  
☐ 7

6. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4  
☐ 4 3 2 1 0  
☐ 1 2 3 4  
☐ 4 3 2 1

7. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
```

```
14
15     ...
16 }
```

- ☐ le terminal affiche x = 5  
☐ le terminal affiche x = 2  
☐ le terminal affiche 5  
☐ le terminal affiche "Faux"

8. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction déclarée mais non définie  
☐ une directive préprocesseur #include manquante  
☐ une fonction appelée avant sa déclaration

9. Pour déclarer une fonction **saisie\_utilisateur** qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ void saisie\_utilisateur(char c);  
☐ saisie\_utilisateur(scanf(%d));  
☐ int saisie\_utilisateur();  
☐ void saisie\_utilisateur(int n);

10. Quel est l'opérateur de différence en C :

- ☐ ≠  
☐ !  
☐ !=  
☐ <>

11. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 1
- ☐ 5

12. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`

13. Un fichier source est :

- ☐ un document de référence du système
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un document qui doit être protégé
- ☐ un document illisible pour les humains
- ☐ un fichier texte qui sera traduit en instructions processeur

14. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration

15. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`
- ☐ `a<=n<=b`
- ☐ `(a<n) || (n>b)`
- ☐ `(n<=a) && (n<=b)`

16. Si cette erreur apparaît à la compilation :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ une faute de frappe dans un appel de fonction
- ☐ une variable non déclarée
- ☐ un caractère interdit en C
- ☐ une directive préprocesseur `#include` manquante

17. Un bit est :

- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur
- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme

18. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas

19. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`

20. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le `main`
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque
- ☐ alphabétique

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- Vous utilisez une boucle `while` quand :
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous avez déjà fait un `for` dans le même programme principal
- Sous unix (ou linux), la commande `ls` permet de :
  - ☐ afficher la liste de fichiers contenus dans un répertoire
  - ☐ compiler un programme
  - ☐ voir des clips musicaux
  - ☐ afficher le contenu d'un fichier texte
- Quel est le problème d'un programme comportant les lignes suivantes ?
 

```
while (1)
{
    printf("coucou\n");
}
```

  - ☐ il n'affiche rien
  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il ne compile pas
  - ☐ il comporte une boucle infinie
- Sur unix (ou linux), la commande `mkdir` permet de :
  - ☐ créer un fichier texte
  - ☐ ouvrir un fichier texte
  - ☐ créer un répertoire
  - ☐ changer de répertoire courant
- Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
  - ☐ `char 'c';`
  - ☐ `char c;`
  - ☐ `int char;`
  - ☐ `char "c";`

- Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 0 0 0 1 1 1
- ☐ 1 2 1 2 3
- ☐ 0 1 0 1 0 1 0 1

- Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`
- ☐ `n = pgcd(int p, int q);`
- ☐ `n = pgcd(n, 3);`
- ☐ `int pgcd(2);`

- Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 4
- ☐ 0
- ☐ 8
- ☐ 16

- Le langage C est un langage

- ☐ compilé
- ☐ composé
- ☐ lu, écrit, parlé
- ☐ interprété

- Quels calculs peut-on programmer en programmation structurée ?

- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ certains programmes sont de vrais plats de spaghetti

- Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur
- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `printf("Valeur de n ? %d\n", n);`

- Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`

- Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`
- ☐ `void exposant(double x^n);`
- ☐ `double exposant(double x, int n);`

- Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`
- ☐ `int factorielle(double n);`

15. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou référence indéfinie vers « printf »`
- ☐ l'analyse sémantique
  - ☐ l'analyse harmonique
  - ☐ l'analyse des entrées clavier
  - ☐ l'édition de liens
16. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :
- ☐ `n = carre(n);`
  - ☐ `int carre(2);`
  - ☐ `n = carre(int n);`
  - ☐ `int n = carre();`
17. L'ordonnancement par tourniquet permet :
- ☐ de doubler la mémoire disponible
  - ☐ de ne pas perdre de temps avec la commutation de contexte

- ☐ d'entretenir l'illusion que les processus tournent en parallèle
  - ☐ d'afficher des ronds colorés à l'écran
18. Le code suivant :
- ```
int i;  
for (i = 8; i > 0; i = i - 2)  
{  
    printf("%d ", i);  
}  
printf("\n");
```
- affichera :
- ☐ 8 6 4 2
  - ☐ 8 6 4 2 0
  - ☐ 8 2
  - ☐ 0 2 4 6 8
19. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :
- ```
int a = 0;  
int b = 20;
```

```
int n;  
scanf("%d", &n);  
while(cond)  
{  
    scanf("%d", &n);  
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
  - ☐ `(a<n) || (n>b)`
  - ☐ `(a<=n) && (n<=b)`
  - ☐ `(n<=a) && (n<=b)`
20. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ dans lequel vous avez déclaré ces fonction
  - ☐ un ordre quelconque
  - ☐ alphabétique
  - ☐ dans lequel ces fonctions sont appelées dans le `main`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un fichier source est :

- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document qui doit être protégé
- ☐ un document illisible pour les humains
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document de référence du système

2. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Écrire des données sur le disque dur
- ☐ Arriver à l'heure en cours

3. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `new TP4`
- ☐ `yppasswd`
- ☐ `mkdir TP4`
- ☐ `kwrite TP4`

4. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien

5. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ `A && B`
- ☐ `(!A || B)`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `!(!A || B) == (A && !B)`

6. Le type des réels en C est :

- ☐ `char`
- ☐ `real`
- ☐ `double`
- ☐ `int`

7. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction

8. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1

9. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ `j = %d`
- ☐ `j = 5`
- ☐ `j = 4`
- ☐ `j = 0`

10. Sous unix (ou linux), la commande `cd` permet de :

- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ jouer de la musique
- ☐ changer de répertoire courant
- ☐ détruire un fichier
- ☐ ouvrir un bureau partagé (common desktop)

11. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6 8
- ☐ 0 2 4 6
- ☐ 0 1 2 3 4 5 6

12. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

13. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `scanf("%d", &n);`
- ☐ un débogueur
- ☐ `printf("Valeur de n ? %g\n", n);`

14. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(2/3);`

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`

15. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal

16. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(double n);`
- ☐ `int factorielle();`
- ☐ `int factorielle(int x);`
- ☐ `struct int factorielle(int n);`

17. Une variable booléenne est un variable :

- ☐ jamais nulle
- ☐ à laquelle une valeur vient d'être affectée

- ☐ qui est vraie ou fausse
- ☐ réelle positive
- ☐ NaN (not a number, qui n'est pas un nombre)

18. Si cette erreur apparaît à la compilation :  
**Undefined symbols : "\_printf" ou référence indéfinie vers < printf > que doit-on chercher dans le programme ?**

- ☐ une variable non déclarée
- ☐ une directive préprocesseur `#include` manquante
- ☐ une faute de frappe dans un appel de fonction
- ☐ un caractère interdit en C

19. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `double afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `void afficher_menu();`
- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu();`

20. Le langage C est un langage

- ☐ composé
- ☐ compilé
- ☐ interprété
- ☐ lu, écrit, parlé

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Une variable booléenne est un variable :

- ☐ à laquelle une valeur vient d'être affectée
- ☐ qui est vraie ou fausse
- ☐ jamais nulle
- ☐ réelle positive
- ☐ NaN (not a number, qui n'est pas un nombre)

2. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ (!A || B)
- ☐ A && B
- ☐ !(A || B) == (A && !B)
- ☐ (A == TRUE) && (B == TRUE)

3. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 8 6 4 2 0
- ☐ 8 6 4 2
- ☐ 8 2

4. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5

5. Quel est l'opérateur de différence en C :

- ☐ !
- ☐ <>
- ☐ !=
- ☐ ≠

6. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ Arriver à l'heure en cours
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ transporter les processus du tourniquet au processeur

7. Un registre du processeur est :

- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ un composant qui contient la liste des fichiers du système

8. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui ne sera pas détectée

9. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses champs
- ☐ ses chants
- ☐ ses blocs

10. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Mineur Majeur
- ☐ Majeur
- ☐ rien

11. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ x = racine(2/3);
- ☐ x - 1 = racine(x);
- ☐ x = racine(racine(x)\*racine(x));
- ☐ x = racine(x \* x) - racine(x);

12. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`

13. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur

14. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 3

15. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche \*\*\*\*
- ☐ le programme affiche x
- ☐ la variable x vaut 16
- ☐ la variable x vaut  $-\frac{1}{2}$

16. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 1 2 3 4

17. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int y, int x);`
- ☐ `void pgcd(int x, int y);`

18. Le type des réels en C est :

- ☐ `char`
- ☐ `double`
- ☐ `real`
- ☐ `int`

19. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(int p, int q);`
- ☐ `int pgcd(2);`
- ☐ `n = pgcd(n, 3);`
- ☐ `int n = pgcd();`

20. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `init`
- ☐ `begin`
- ☐ `include`
- ☐ `main`



**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 4  
☐ j = %d  
☐ j = 0  
☐ j = 5

2. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ (a.n == b.n) && (a.x == b.x)  
☐ a == b  
☐ a = b  
☐ a{n, x} == b{n, x}

3. L'ordonnancement par tourniquet permet :

- ☐ d'entretenir l'illusion que les processus tournent en parallèle  
☐ de ne pas perdre de temps avec la commutation de contexte  
☐ de doubler la mémoire disponible  
☐ d'afficher des ronds colorés à l'écran

4. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 0  
☐ 7  
☐ 5

5. Pour déclarer une fonction pgcd qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ void pgcd(int x, int y);  
☐ int pgcd(int x, y);  
☐ int pgcd(int x, int x);  
☐ int pgcd(int y, int x);

6. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 5  
☐ 3  
☐ 4  
☐ 101

7. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ char "c";  
☐ int char;  
☐ char 'c';  
☐ char c;

8. Si **carre** est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que n est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ n = carre(n);  
☐ n = carre(int n);  
☐ int carre(2);  
☐ int n = carre();

9. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ la variable x vaut 5 et la variable y vaut 0  
☐ le programme affiche "Faux"  
☐ la variable x vaut 0  
☐ la variable y vaut 5

10. Le type des réels en C est :

- ☐ real  
☐ int  
☐ double  
☐ char

11. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ retourner un bloc  
☐ répéter un bloc tant qu'une condition est vérifiée  
☐ sélectionner entre deux blocs à l'aide d'une condition  
☐ mettre les blocs en séquence les uns à la suite des autres

12. Sur unix (ou linux), la commande **mkdir** permet de :

- ☐ changer de répertoire courant  
☐ créer un répertoire  
☐ ouvrir un fichier texte  
☐ créer un fichier texte

13. Vous utilisez une boucle `while` quand :
- ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous avez déjà fait un `for` dans le même programme principal
14. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :
- ☐ `kwrite` TP4
  - ☐ `mkdir` TP4
  - ☐ `new` TP4
  - ☐ `yppasswd`
15. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :
- ☐ `n = pgcd(int p, int q);`
  - ☐ `int pgcd(2);`
  - ☐ `int n = pgcd();`
  - ☐ `n = pgcd(n, 3);`
16. Pour compiler un programme `prog.c`, on utilise la ligne de commande :
- ☐ `gcc prog.exe -Wall -o prog.c`
  - ☐ `gcc prog.c -o -Wall prog.exe`
  - ☐ `gcc -Wall prog.c -o prog.exe`
  - ☐ `gcc -Wall prog.exe -o prog.c`

17. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 4
- ☐ 3

18. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`

19. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`  
, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration

- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction

20. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a{n, x} == b{n, x}
- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a = b
- ☐ a == b

2. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max',**  
que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur **#include** manquante

3. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

- ☐ printf("x=%d et y=%d\n",x,y);
- ☐ printf("x=%d et y=%d\n",x y);
- ☐ printf("x=%d et y=%d\n",x,y);
- ☐ printf("x=%x et y=%y\n");

4. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 3 0 1 2
- ☐ 0 1 2 0 1 2 3

5. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 4 3 2 1

6. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ int %d;
- ☐ loop i;
- ☐ int k;
- ☐ int loop n;

7. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque
- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main

8. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

9. Pour déclarer une procédure **afficher\_date** qui prend en argument un **struct date\_s** et affiche le contenu du struct, on écrit :

- ☐ void afficher\_date(struct date\_s d);
- ☐ void afficher\_date(date\_s d);
- ☐ struct date\_s afficher\_date(struct date\_s d);
- ☐ int afficher\_date(date\_s d);

10. Lorsqu'un programme utilise **printf** ou **scanf** il faut qu'il contienne l'instruction préprocesseur :

- ☐ #appart <stdlib.h>
- ☐ #include <studlib.h>
- ☐ #include <stdio.h>
- ☐ #include <studio.h>

11. Un registre du processeur est :

- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ un composant qui contient la liste des fichiers du système

12. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 0
- ☐ 10
- ☐ 15

13. Un bit est :

- ☐ un battement d'horloge processeur
- ☐ un chiffre binaire (0 ou 1)
- ☐ l'instruction qui met fin à un programme
- ☐ la longueur d'un mot mémoire

14. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 6
- ☐ 1
- ☐ 42
- ☐ 0

15. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5

16. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses champs
- ☐ ses chants
- ☐ ses blocs

17. Au début de la fonction main() on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A

☐ C

☐ b

☐ B

18. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(int n);`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `saisie_utilisateur(scanf(%d));`

19. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

20. Le langage C est un langage

- ☐ interprété
- ☐ compilé
- ☐ composé
- ☐ lu, écrit, parlé

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

☐ `int afficher_date(date_s d);`  
☐ `void afficher_date(struct date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`  
☐ `void afficher_date(date_s d);`

2. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

☐ Majeur  
☐ Mineur  
☐ Majeur  
☐ Mineur  
☐ rien

3. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

☐ le terminal affiche `x = 5`  
☐ le terminal affiche 5  
☐ le terminal affiche `x = 2`  
☐ le terminal affiche "Faux"

4. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

☐ `printf("x=%d et y=%d\n",x,y);`  
☐ `printf("x=%x et y=%y\n");`  
☐ `printf("x=%d et y=%d\n,x,y");`  
☐ `printf("x=%d et y=%d\n",x y);`

5. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

☐ `int carre(2);`  
☐ `n = carre(int n);`  
☐ `n = carre(n);`  
☐ `int n = carre();`

6. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

☐ `x - 1 = racine(x);`  
☐ `x = racine(racine(x)*racine(x));`  
☐ `x = racine(x * x) - racine(x);`  
☐ `x = racine(2/3);`

7. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

☐ un débogueur  
☐ `printf("Valeur de n ? %d\n", n);`  
☐ `printf("Valeur de n ? %g\n", n);`  
☐ `scanf("%d", &n);`

8. Sur unix (ou linux), la commande `mkdir` permet de :

☐ créer un répertoire  
☐ créer un fichier texte  
☐ ouvrir un fichier texte  
☐ changer de répertoire courant

9. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

☐ ABCDEF  
☐ A  
☐ cccccc  
☐ i

10. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

☐ `a<=n<=b`  
☐ `(a<=n) && (n<=b)`  
☐ `(n<=a) && (n<=b)`  
☐ `(a<n) || (n>b)`

11. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

☐ `n = pgcd(n, 3);`  
☐ `n = pgcd(int p, int q);`  
☐ `int n = pgcd();`  
☐ `int pgcd(2);`

12. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1
- ☐ 0 0 1 1 2 2
- ☐ 0 1 2 0 1 2
- ☐ 1 2 3 1 2

13. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`

14. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

15. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ la variable `x` vaut 5 et la variable `y` vaut 0
- ☐ la variable `x` vaut 0
- ☐ la variable `y` vaut 5
- ☐ le programme affiche "Faux"

16. Un bit est :

- ☐ la longueur d'un mot mémoire
- ☐ un battement d'horloge processeur
- ☐ un chiffre binaire (0 ou 1)
- ☐ l'instruction qui met fin à un programme

17. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 4
- ☐ 3

18. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 8 2
- ☐ 8 6 4 2
- ☐ 8 6 4 2 0

19. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ rien

20. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

- Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
  - ☐ `char c;`
  - ☐ `char "c";`
  - ☐ `int char;`
  - ☐ `char 'c';`
- Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :
  - ☐ `char afficher_menu printf("menu");`
  - ☐ `double afficher_menu();`
  - ☐ `int afficher_menu();`
  - ☐ `int afficher_menu(int char);`
  - ☐ `void afficher_menu();`
- Sur unix (ou linux), la commande `mkdir` permet de :
  - ☐ créer un fichier texte
  - ☐ ouvrir un fichier texte
  - ☐ changer de répertoire courant
  - ☐ créer un répertoire
- Le code suivant :
 

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

 affichera :
  - ☐ 0 2 4 6 8
  - ☐ 8 6 4 2 0
  - ☐ 8 6 4 2
  - ☐ 8 2
- Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction appelée avant sa déclaration
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction déclarée mais non définie

- Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
  - ☐ dans lequel ces fonctions sont appelées dans le main
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ alphabétique
  - ☐ un ordre quelconque
- L'écriture 101 en binaire correspond au nombre naturel :
  - ☐ 101
  - ☐ 3
  - ☐ 4
  - ☐ 5
- Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :
  - ☐ `int n = pgcd();`
  - ☐ `n = pgcd(int p, int q);`
  - ☐ `n = pgcd(n, 3);`
  - ☐ `int pgcd(2);`
- On considère deux variables booléennes A et B initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?
  - ☐ `A && B`
  - ☐ `(!A || B)`
  - ☐ `!(!A || B) == (A && !B)`
  - ☐ `(A == TRUE) && (B == TRUE)`
- Le langage C est un langage
  - ☐ compilé
  - ☐ composé
  - ☐ interprété
  - ☐ lu, écrit, parlé

- Un enregistrement permet de grouper plusieurs valeurs dans :
  - ☐ ses cases
  - ☐ ses blocs
  - ☐ ses champs
  - ☐ ses chants
- Avant de faire appel à une fonction il est nécessaire de :
  - ☐ l'avoir déclarée et définie
  - ☐ avoir défini une constante symbolique de la taille de cette fonction
  - ☐ l'avoir déclarée
  - ☐ l'avoir définie
- Soit la fonction `g` définie par :
 

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

 Alors l'expression `g(0)` prendra la valeur :
  - ☐ 0
  - ☐ 7
  - ☐ 5
- Le code suivant :
 

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

 affichera :
  - ☐ 0 1 2 3
  - ☐ 4 3 2 1 0
  - ☐ 0 1 2 3 4
  - ☐ 4 3 2 1

15. Si x est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 0
- ☐ 0.5
- ☐ 1.5
- ☐ 1

16. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse lexicale
- ☐ analyse sémantique
- ☐ analyse syntaxique

17. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
```

```
    printf("%d ", i);
}
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1

18. Le type des réels en C est :

- ☐ char
- ☐ double
- ☐ real
- ☐ int

19. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int tab[] = 5;`

- ☐ `int toto[taille=5];`
- ☐ `int[] new tableau(5);`
- ☐ `int toto[5];`
- ☐ `char tableau[5];`

20. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a{n, x} == b{n, x}
- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a == b
- ☐ a = b

2. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 8 6 4 2
- ☐ 8 6 4 2 0
- ☐ 8 2

3. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ int tab[] = 5;
- ☐ int toto[taille=5];
- ☐ int[] new tableau(5);
- ☐ int toto[5];
- ☐ char tableau[5];

4. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 4
- ☐ 5
- ☐ 1
- ☐ 0

5. Quel est l'opérateur de différence en C :

- ☐ !
- ☐ !=
- ☐ <>
- ☐ ≠

6. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ loop i;
- ☐ int loop n;
- ☐ int k;
- ☐ int %d;

7. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ char "c";
- ☐ char 'c';
- ☐ int char;
- ☐ char c;

8. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 7
- ☐ 0
- ☐ 5

9. Si **carre** est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que n est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ n = carre(int n);
- ☐ int n = carre();
- ☐ n = carre(n);
- ☐ int carre(2);

10. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

11. Sur unix (ou linux), la commande `mkdir` permet de :
- ☐ changer de répertoire courant
  - ☐ créer un fichier texte
  - ☐ créer un répertoire
  - ☐ ouvrir un fichier texte
12. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`, que doit-on chercher dans le programme ?
- ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction appelée avant sa déclaration
  - ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
13. Les lignes
- ```
int i;  
int x=0;  
for(i=0,i<5,i=i+1)  
{  
    x=x+1;  
}
```
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
  - ☐ ne comportent aucune erreur
  - ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
  - ☐ comportent une erreur qui ne sera pas détectée

14. Quels calculs peut-on programmer en programmation structurée ?
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
  - ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
  - ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
  - ☐ certains programmes sont de vrais plats de spaghetti
15. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :
- ☐ `int mccarthy(int 2);`
  - ☐ `x = mccarthy(n);`
  - ☐ `n = mccarthy();`
  - ☐ `n = mccarthy(p, q);`
16. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :
- ☐ `printf("x=%d et y=%d\n",x y);`
  - ☐ `printf("x=%x et y=%y\n");`
  - ☐ `printf("x=%d et y=%d\n,x,y");`
  - ☐ `printf("x=%d et y=%d\n",x,y);`
17. Au début de la fonction `main()` on place le code :
- ```
char b = 'A';  
b = b + 2;  
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C
  - ☐ B
  - ☐ A
  - ☐ b
18. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :
- ☐ `struct date_s afficher_date(struct date_s d);`
  - ☐ `void afficher_date(struct date_s d);`
  - ☐ `int afficher_date(date_s d);`
  - ☐ `void afficher_date(date_s d);`
19. Un programme en langage C doit comporter une et une seule définition de la fonction :
- ☐ `main`
  - ☐ `include`
  - ☐ `init`
  - ☐ `begin`
20. L'écriture 101 en binaire correspond au nombre naturel :
- ☐ 5
  - ☐ 101
  - ☐ 4
  - ☐ 3

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = 0  
☐ j = %d  
☐ j = 4  
☐ j = 5

2. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2  
☐ 0 1 0 1 0 1 0 1  
☐ 0 0 0 1 1 1  
☐ 1 2 1 2 3

3. Vous utilisez une boucle **while** quand :

- ☐ vous n'avez pas déclaré de fonction  
☐ l'incrément de la variable de boucle n'est pas 1

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ vous avez déjà fait un **for** dans le même programme principal

4. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a = b  
☐ a{n, x} == b{n, x}  
☐ a == b  
☐ (a.n == b.n) && (a.x == b.x)

5. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 10  
☐ 6  
☐ 15  
☐ 0

6. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs  
☐ ses cases  
☐ ses blocs  
☐ ses chants

7. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire  
☐ transporter les processus du tourniquet au processeur  
☐ Arriver à l'heure en cours  
☐ Écrire des données sur le disque dur

8. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction **main()**, alors on peut écrire en début de **main()** :

- ☐ toto\_s n, x;  
☐ struct toto\_s toto;  
☐ int struct toto\_s = {3, -1e10};  
☐ int toto.n = 3;  
☐ toto\_s struct z = {3, 0.5};

9. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir définie  
☐ l'avoir déclarée  
☐ l'avoir déclarée et définie  
☐ avoir défini une constante symbolique de la taille de cette fonction

10. Pour compiler un programme **prog.c**, on utilise la ligne de commande :

- ☐ gcc prog.c -o -Wall prog.exe  
☐ gcc -Wall prog.exe -o prog.c  
☐ gcc prog.exe -Wall -o prog.c  
☐ gcc -Wall prog.c -o prog.exe

11. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ int factorielle();  
☐ int factorielle(double n);  
☐ struct int factorielle(int n);  
☐ int factorielle(int x);

12. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 4
- ☐ 1

13. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`
- ☐ `void exposant(double x^n);`
- ☐ `double exposant(double x, int n);`
- ☐ `int exposant(double n, int x);`

14. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n",x,y);`

15. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(date_s d);`

16. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 0.5
- ☐ 1.5
- ☐ 1
- ☐ 0

17. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ en parallèle, chacun dans un registre
- ☐ tous ensemble
- ☐ tour à tour, un petit peu à chaque fois

18. Quel est l'opérateur de différence en C :

- ☐ `≠`
- ☐ `!`
- ☐ `!=`
- ☐ `<>`

19. Le langage C est un langage

- ☐ compilé
- ☐ composé
- ☐ interprété
- ☐ lu, écrit, parlé

20. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche `****`
- ☐ le programme affiche `x`
- ☐ la variable `x` vaut  $-\frac{1}{2}$
- ☐ la variable `x` vaut 16

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose
- ☐ la case mémoire 8 contiendra 16

2. Si  $x$  est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1.5
- ☐ 0
- ☐ 1
- ☐ 0.5

3. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable  $x$  vaut 0
- ☐ la variable  $x$  vaut 5 et la variable  $y$  vaut 0
- ☐ le programme affiche "Faux"
- ☐ la variable  $y$  vaut 5

4. Une variable booléenne est un variable :

- ☐ qui est vraie ou fausse
- ☐ réelle positive
- ☐ à laquelle une valeur vient d'être affectée
- ☐ jamais nulle
- ☐ NaN (not a number, qui n'est pas un nombre)

5. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou

6. Un bit est :

- ☐ un chiffre binaire (0 ou 1)
- ☐ l'instruction qui met fin à un programme
- ☐ la longueur d'un mot mémoire
- ☐ un battement d'horloge processeur

7. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 0
- ☐ 4
- ☐ 16

8. Sous unix (ou linux), la commande `ls` permet de :

- ☐ compiler un programme
- ☐ voir des clips musicaux
- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ afficher le contenu d'un fichier texte

9. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop
- ☐ une accolade en trop
- ☐ un point-virgule manquant
- ☐ une accolade manquante

10. L'ordonnancement par tourniquet permet :

- ☐ de doubler la mémoire disponible
- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'afficher des ronds colorés à l'écran

11. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`

12. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses blocs
- ☐ ses cases
- ☐ ses chants

13. Le type des réels en C est :

- ☐ `double`
- ☐ `int`
- ☐ `real`
- ☐ `char`

14. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
- ☐ `char c;`
  - ☐ `char 'c';`
  - ☐ `int char;`
  - ☐ `char "c";`
15. Vous utilisez une boucle `while` quand :
- ☐ vous n'avez pas déclaré de fonction
  - ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
16. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :
- ☐ `saisie_utilisateur scanf(%d);`
  - ☐ `int saisie_utilisateur();`
  - ☐ `void saisie_utilisateur(char c);`
  - ☐ `void saisie_utilisateur(int n);`
17. Le langage C est un langage
- ☐ compilé
  - ☐ composé
  - ☐ interprété
  - ☐ lu, écrit, parlé

18. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

19. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`
- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`

20. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 20
- ☐ 16
- ☐ 6
- ☐ 3

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 16

2. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(a<n) || (n>b)`

3. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C
- ☐ b
- ☐ A
- ☐ B

4. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

5. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `A && B`
- ☐ `(!A || B)`
- ☐ `!(A || B) == (A && !B)`
- ☐ `(A == TRUE) && (B == TRUE)`

6. Après exécution jusqu'à la ligne 15 du programme C :

```
10  ...
11  int main() {
12      int x = 5;
13
14      x = 3 * x + 1;
15
16      ...
17  }
```

- ☐ le programme affiche `****`
- ☐ la variable `x` vaut 16
- ☐ le programme affiche `x`
- ☐ la variable `x` vaut  $-\frac{1}{2}$

7. Un registre du processeur est :

- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ un composant qui contient la liste des fichiers du système

8. Vous utilisez une boucle `while` quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un `for` dans le même programme principal

9. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

10. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un répertoire
- ☐ ouvrir un fichier texte
- ☐ changer de répertoire courant
- ☐ créer un fichier texte

11. Le langage C est un langage

- ☐ composé
- ☐ lu, écrit, parlé
- ☐ interprété
- ☐ compilé

12. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define N = 3`
- ☐ `#define taille = 3`
- ☐ `#define N 3`
- ☐ `#define taille = N`

13. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 4
- ☐ j = 0
- ☐ j = 5
- ☐ j = %d

14. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque

15. Quel est le problème d'un programme comportant les lignes suivantes?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou

16. Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu();`
- ☐ `void afficher_menu();`
- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu(int char);`
- ☐ `double afficher_menu();`

17. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `int exposant(double n, int x);`
- ☐ `void exposant(double x^n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `double exposant(double x, int n);`

18. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 2
- ☐ 8 6 4 2
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2 0

19. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur

20. Le type des réels en C est :

- ☐ `char`
- ☐ `int`
- ☐ `double`
- ☐ `real`



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `int toto.n = 3;`
- ☐ `struct toto_s toto;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `toto_s n, x;`

2. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti

3. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`

4. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

5. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `yppasswd`
- ☐ `mkdir TP4`
- ☐ `kwrite TP4`
- ☐ `new TP4`

6. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`
- ☐ `n = carre(int n);`
- ☐ `n = carre(n);`
- ☐ `int n = carre();`

7. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours
- ☐ transporter les processus du tourniquet au processeur

8. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut lancer un débogueur
- ☐ qu'il faut indenter le fichier source

9. Un bit est :

- ☐ un battement d'horloge processeur
- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)
- ☐ la longueur d'un mot mémoire

10. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction

11. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max' ,** que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie

12. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 0
- ☐ 10
- ☐ 15

13. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 7

14. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`

15. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ le programme affiche "Faux"
- ☐ la variable `y` vaut 5
- ☐ la variable `x` vaut 0
- ☐ la variable `x` vaut 5 et la variable `y` vaut 0

16. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce `printf` ?

- ☐ `j = 0`
- ☐ `j = 4`
- ☐ `j = 5`
- ☐ `j = %d`

17. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 1 2 1 2 3
- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
- ☐ 0 0 0 1 1 1

18. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<n) || (n>b)`
- ☐ `a<=n<=b`

19. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée
- ☐ l'avoir déclarée et définie
- ☐ l'avoir définie
- ☐ avoir défini une constante symbolique de la taille de cette fonction

20. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8
- ☐ 111
- ☐ 7
- ☐ 3

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose
- ☐ la case mémoire 8 contiendra 16

2. Le langage C est un langage

- ☐ lu, écrit, parlé
- ☐ composé
- ☐ compilé
- ☐ interprété

3. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 1 2 1 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 0 0 1 1 1

4. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 7
- ☐ 5
- ☐ 0

5. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ Arriver à l'heure en cours
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ transporter les processus du tourniquet au processeur

6. Vous utilisez une boucle while quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un for dans le même programme principal

7. Quels calculs peut-on programmer en programmation structurée ?

- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine

8. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique
- ☐ un ordre quelconque

9. Au début de la fonction main() on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A
- ☐ ABCDEF
- ☐ cccccc
- ☐ i

10. Au début de la fonction main() on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b
- ☐ B
- ☐ A
- ☐ C

11. Après la déclaration : int mccarthy(int n);, il est correct d'écrire :

- ☐ n = mccarthy();
- ☐ n = mccarthy(p, q);
- ☐ int mccarthy(int 2);
- ☐ x = mccarthy(n);

12. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a = b
- ☐ a == b
- ☐ a{n, x} == b{n, x}
- ☐ (a.n == b.n) && (a.x == b.x)

13. Si pgcd est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ int pgcd(2);
- ☐ n = pgcd(int p, int q);
- ☐ int n = pgcd();
- ☐ n = pgcd(n, 3);

14. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou

15. Si racine est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ x = racine(x \* x) - racine(x);
- ☐ x = racine(racine(x)\*racine(x));
- ☐ x - 1 = racine(x);
- ☐ x = racine(2/3);

16. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 5
- ☐ j = 0
- ☐ j = 4
- ☐ j = %d

17. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

Undefined symbols : "\_printf" ou  
référence indéfinie vers « printf »

- ☐ l'analyse sémantique

- ☐ l'analyse harmonique
- ☐ l'analyse des entrées clavier
- ☐ l'édition de liens

18. Pour compiler un programme prog.c, on utilise la ligne de commande :

- ☐ gcc -Wall prog.c -o prog.exe
- ☐ gcc -Wall prog.exe -o prog.c
- ☐ gcc prog.exe -Wall -o prog.c
- ☐ gcc prog.c -o -Wall prog.exe

19. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 0
- ☐ la variable x vaut 5 et la variable y vaut 0

20. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char "c";
- ☐ int char;
- ☐ char c;
- ☐ char 'c';

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 15  
☐ 6  
☐ 0  
☐ 10

2. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 1  
☐ 5  
☐ 0  
☐ 4

3. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche x = 5  
☐ le terminal affiche "Faux"  
☐ le terminal affiche x = 2  
☐ le terminal affiche 5

4. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ (A == TRUE) && (B == TRUE)  
☐ A && B  
☐ (!A || B)  
☐ !(A || B) == (A && !B)

5. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8  
☐ 16  
☐ 4  
☐ 0

6. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8  
☐ 0 2 4 6  
☐ 0 1 2 3 4 5 6  
☐ 0 1 2 3 4 5 6 7

7. Si pgcd est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ int n = pgcd();  
☐ n = pgcd(int p, int q);  
☐ n = pgcd(n, 3);  
☐ int pgcd(2);

8. Au début de la fonction main() on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C  
☐ B  
☐ b  
☐ A

9. Lorsqu'un programme utilise printf ou scanf il faut qu'il contienne l'instruction préprocesseur :

- ☐ #appart <stdlib.h>  
☐ #include <studio.h>  
☐ #include <stdio.h>  
☐ #include <studlib.h>

10. Pour déclarer une procédure afficher\_date qui prend en argument un struct date\_s et affiche le contenu du struct, on écrit :

- ☐ void afficher\_date(date\_s d);  
☐ void afficher\_date(struct date\_s d);  
☐ int afficher\_date(date\_s d);  
☐ struct date\_s afficher\_date(struct date\_s d);

11. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

12. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses blocs
- ☐ ses cases
- ☐ ses champs

13. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `begin`
- ☐ `init`
- ☐ `main`
- ☐ `include`

14. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché?

- ☐ 0 1 2 0 1 2
- ☐ 0 0 1 1 2 2
- ☐ 1 2 3 1 2
- ☐ 0 1 0 1 0 1

15. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements

16. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(double n);`
- ☐ `int factorielle();`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(int x);`

17. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le `main`
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque

18. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Majeur
- ☐ Mineur

19. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

20. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`
- ☐ `n = mccarthy(p, q);`
- ☐ `int mccarthy(int 2);`
- ☐ `x = mccarthy(n);`

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 5 et la variable y vaut 0
- ☐ la variable y vaut 5
- ☐ la variable x vaut 0

2. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a == b
- ☐ a = b
- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a{n, x} == b{n, x}

3. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ int factorielle(int x);
- ☐ struct int factorielle(int n);
- ☐ int factorielle();
- ☐ int factorielle(double n);

4. Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :

- ☐ double afficher\_menu();
- ☐ void afficher\_menu();
- ☐ int afficher\_menu();
- ☐ char afficher\_menu(sprintf("menu"));
- ☐ int afficher\_menu(int char);

5. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 0
- ☐ j = %d
- ☐ j = 5
- ☐ j = 4

6. Un bit est :

- ☐ l'instruction qui met fin à un programme
- ☐ la longueur d'un mot mémoire
- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur

7. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ begin
- ☐ include
- ☐ main
- ☐ init

8. Une variable booléenne est un variable :

- ☐ jamais nulle
- ☐ à laquelle une valeur vient d'être affectée
- ☐ réelle positive
- ☐ qui est vraie ou fausse
- ☐ NaN (not a number, qui n'est pas un nombre)

9. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses chants
- ☐ ses champs
- ☐ ses cases

10. Si **factorielle** est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ int factorielle(int 2);
- ☐ n = factorielle(p, q);
- ☐ n = factorielle();
- ☐ printf("%d", factorielle(n));

11. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction **main()**, alors on peut écrire en début de **main()** :

- ☐ int struct toto\_s = {3, -1e10};
- ☐ toto\_s n, x;
- ☐ toto\_s struct z = {3, 0.5};
- ☐ int toto.n = 3;
- ☐ struct toto\_s toto;

12. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression **f(0)** prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

13. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 3 0 1 2

14. Le langage C est un langage

- ☐ interprété
- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ composé

15. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`
- ☐ `n = mccarthy(p, q);`
- ☐ `x = mccarthy(n);`
- ☐ `int mccarthy(int 2);`

16. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%x et y=%y\n");`

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Mineur
- ☐ Majeur
- ☐ Majeur
- ☐ Mineur

18. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2 0
- ☐ 8 6 4 2
- ☐ 0 2 4 6 8
- ☐ 8 2

19. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(n<=a) && (n<=b)`
- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(a<n) || (n>b)`

20. Un registre du processeur est :

- ☐ un composant qui contient la liste des fichiers du système
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie  
☐ il n'affiche rien  
☐ il risque d'afficher bonjour à la place de coucou  
☐ il ne compile pas

2. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0  
☐ 4  
☐ 3

3. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ rien  
☐ Majeur  
☐ Mineur  
☐ Mineur  
☐ Majeur

4. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `struct toto_s toto;`  
☐ `toto_s struct z = {3, 0.5};`  
☐ `int struct toto_s = {3, -1e10};`  
☐ `toto_s n, x;`  
☐ `int toto.n = 3;`

5. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8  
☐ 8 6 4 2 0  
☐ 8 2  
☐ 8 6 4 2

6. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc prog.exe -Wall -o prog.c`  
☐ `gcc -Wall prog.c -o prog.exe`  
☐ `gcc -Wall prog.exe -o prog.c`  
☐ `gcc prog.c -o -Wall prog.exe`

7. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale  
☐ analyse sémantique  
☐ analyse harmonique  
☐ analyse syntaxique

8. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche "Faux"  
☐ le terminal affiche `x = 5`  
☐ le terminal affiche 5  
☐ le terminal affiche `x = 2`

9. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique  
☐ comportent une erreur qui sera détectée au cours de l'édition de lien  
☐ ne comportent aucune erreur  
☐ comportent une erreur qui ne sera pas détectée

10. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `int saisie_utilisateur();`  
☐ `void saisie_utilisateur(char c);`  
☐ `void saisie_utilisateur(int n);`  
☐ `saisie_utilisateur(scanf(%d));`

11. Un fichier source est :

- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur  
☐ un document de référence du système  
☐ un document qui doit être protégé  
☐ un document illisible pour les humains  
☐ un fichier texte qui sera traduit en instructions processeur

12. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :
- ☐ `int factorielle(int x);`
  - ☐ `int factorielle(double n);`
  - ☐ `int factorielle();`
  - ☐ `struct int factorielle(int n);`
13. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?
- ☐ une fonction déclarée mais non définie
  - ☐ une directive préprocesseur **#include** manquante
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction appelée avant sa déclaration
14. Le bus système sert à :
- ☐ Écrire des données sur le disque dur
  - ☐ Transférer des données et intructions entre processeur et mémoire
  - ☐ transporter les processus du tourniquet au processeur
  - ☐ Arriver à l'heure en cours
15. Si **factorielle** est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
- ☐ `n = factorielle();`
  - ☐ `int factorielle(int 2);`
  - ☐ `printf("%d", factorielle(n));`
  - ☐ `n = factorielle(p, q);`

16. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
- ☐ une directive préprocesseur **#include** manquante
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction déclarée mais non définie
  - ☐ une fonction appelée avant sa déclaration
17. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```
- ☐ la variable x vaut 0
  - ☐ la variable y vaut 5
  - ☐ le programme affiche "Faux"
  - ☐ la variable x vaut 5 et la variable y vaut 0
18. Après exécution du programme :
- ```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
```

- ```
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```
- ☐ la case mémoire 8 contiendra 16
  - ☐ le terminal affiche 8
  - ☐ le bus explose
  - ☐ la case mémoire 8 contiendra 0
19. Si **carre** est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que n est une variable entière définie et initialisée, il est correct d'écrire :
- ☐ `n = carre(n);`
  - ☐ `int n = carre();`
  - ☐ `int carre(2);`
  - ☐ `n = carre(int n);`
20. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ alphabétique
  - ☐ un ordre quelconque
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ dans lequel ces fonctions sont appelées dans le main

**Barème : 1 point par réponse juste (unique) ; -0,5 point par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

☐ `int afficher_menu(int char);`  
☐ `void afficher_menu();`  
☐ `double afficher_menu();`  
☐ `int afficher_menu();`  
☐ `char afficher_menu(sprintf("menu"));`

2. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

☐ `#include <studio.h>`  
☐ `#appart <stdlib.h>`  
☐ `#include <studlib.h>`  
☐ `#include <stdio.h>`

3. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

☐ 0 1 2 3 4  
☐ 1 2 3 4  
☐ 4 3 2 1  
☐ 4 3 2 1 0

4. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.

☐ `#define taille = 3`  
☐ `#define N = 3`  
☐ `#define taille = N`  
☐ `#define N 3`

5. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`, que doit-on chercher dans le programme ?

☐ une fonction appelée avant sa déclaration  
☐ une directive préprocesseur `#include` manquante  
☐ une fonction déclarée mais non définie  
☐ un désaccord entre la déclaration et la définition d'une fonction

6. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

☐ analyse syntaxique  
☐ analyse lexicale  
☐ analyse sémantique  
☐ analyse harmonique

7. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

☐ `x = mccarthy(n);`  
☐ `n = mccarthy();`  
☐ `int mccarthy(int 2);`  
☐ `n = mccarthy(p, q);`

8. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

☐ `int loop n;`  
☐ `loop i;`  
☐ `int k;`  
☐ `int %d;`

9. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

☐ `f(a,b)=13, a=8, b=5`  
☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=8, a=3, b=5`  
☐ `f(3,5)=8, a=3, b=5`

10. Si cette erreur apparaît à la compilation :  
`Undefined symbols : "_printf" ou référence indéfinie vers « printf »` que doit-on chercher dans le programme ?

☐ une directive préprocesseur `#include` manquante  
☐ une variable non déclarée  
☐ une faute de frappe dans un appel de fonction  
☐ un caractère interdit en C

11. Si cette erreur apparaît à la compilation :  
`erreur: conflicting types for 'max'`, que doit-on chercher dans le programme ?

☐ une directive préprocesseur `#include` manquante  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction appelée avant sa déclaration  
☐ une fonction déclarée mais non définie

12. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

☐ 0.5  
☐ 1.5  
☐ 0  
☐ 1

13. Un enregistrement permet de grouper plusieurs valeurs dans :

☐ ses blocs  
☐ ses champs  
☐ ses cases  
☐ ses chants

14. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 16

15. Si cette erreur apparaît à la compilation :

**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade manquante
- ☐ un point-virgule manquant
- ☐ une accolade en trop
- ☐ un point-virgule en trop

16. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il comporte une boucle infinie

17. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(int n);`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`
- ☐ `saisie_utilisateur(scanf(%d));`

18. Une variable booléenne est une variable :

- ☐ qui est vraie ou fausse
- ☐ à laquelle une valeur vient d'être affectée
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ jamais nulle
- ☐ réelle positive

19. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
}
```

```
}
return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

20. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 1 2 1 2 3
- ☐ 0 0 0 1 1 1
- ☐ 0 1 0 1 0 1 0 1

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`  
, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction appelée avant sa déclaration  
☐ une fonction déclarée mais non définie  
☐ une directive préprocesseur `#include` manquante

2. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`  
☐ `double exposant(double x, int n);`  
☐ `int exposant(double n, int x);`  
☐ `exposant(double x, int n, int r);`

3. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3  
☐ 4  
☐ 0

4. Sous unix (ou linux), la commande `ls` permet de :

- ☐ afficher la liste de fichiers contenus dans un répertoire  
☐ compiler un programme  
☐ voir des clips musicaux  
☐ afficher le contenu d'un fichier texte

5. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie  
☐ il n'affiche rien  
☐ il ne compile pas  
☐ il risque d'afficher bonjour à la place de coucou

6. Quel est l'opérateur de différence en C :

- ☐ `!`  
☐ `!=`  
☐ `≠`  
☐ `<>`

7. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un répertoire  
☐ ouvrir un fichier texte  
☐ créer un fichier texte  
☐ changer de répertoire courant

8. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur  
Majeur  
☐ rien  
☐ Majeur  
☐ Mineur

9. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur  
☐ Mineur  
Majeur  
☐ Mineur  
☐ rien

10. Une variable booléenne est un variable :

- ☐ réelle positive  
☐ à laquelle une valeur vient d'être affectée  
☐ NaN (not a number, qui n'est pas un nombre)  
☐ qui est vraie ou fausse  
☐ jamais nulle

11. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants  
☐ ses cases  
☐ ses blocs  
☐ ses champs

12. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition cond :

- ☐ (n<=a) && (n<=b)
- ☐ a<=n<=b
- ☐ (a<=n) && (n<=b)
- ☐ (a<n) || (n>b)

13. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`

14. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 20
- ☐ 6
- ☐ 3
- ☐ 16

15. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(double n);`
- ☐ `int factorielle();`

16. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 1 2 3 1 2
- ☐ 0 1 0 1 0 1
- ☐ 0 0 1 1 2 2
- ☐ 0 1 2 0 1 2

17. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4

18. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 7
- ☐ 111
- ☐ 8

19. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours

20. Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :

- ☐ `char afficher_menu(printf("menu"));`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `int afficher_menu();`
- ☐ `double afficher_menu();`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Sur unix (ou linux), la commande `mkdir` permet de :
- ☐ ouvrir un fichier texte
  - ☐ créer un répertoire
  - ☐ changer de répertoire courant
  - ☐ créer un fichier texte

2. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`
  - ☐ `a<=n<=b`
  - ☐ `(a<=n) && (n<=b)`
  - ☐ `(n<=a) && (n<=b)`
3. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :
- ☐ `printf("x=%d et y=%d\n",x,y);`
  - ☐ `printf("x=%d et y=%d\n",x y);`
  - ☐ `printf("x=%x et y=%y\n");`
  - ☐ `printf("x=%d et y=%d\n",x,y);`
4. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ un ordre quelconque
  - ☐ alphabétique
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ dans lequel ces fonctions sont appelées dans le `main`

5. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1
- ☐ 1 2 3 4

6. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 5
- ☐ 0

7. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses chants
  - ☐ ses champs
  - ☐ ses blocs
  - ☐ ses cases

8. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

```
void afficher_date(date_s d);
int afficher_date(date_s d);
void afficher_date(struct date_s d);
struct date_s afficher_date(struct date_s d);
```

9. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

Undefined symbols : "\_printf" ou référence indéfinie vers < printf >

- ☐ l'analyse des entrées clavier
- ☐ l'édition de liens
- ☐ l'analyse sémantique
- ☐ l'analyse harmonique

10. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 20
- ☐ 6
- ☐ 16
- ☐ 3

11. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ la variable `x` vaut 5 et la variable `y` vaut 0
- ☐ la variable `y` vaut 5
- ☐ la variable `x` vaut 0
- ☐ le programme affiche "Faux"

12. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

13. Vous utilisez une boucle **while** quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un **for** dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

14. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3

15. Dans la commande gcc, l'option **-Wall** signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur
- ☐ qu'il faut indenter le fichier source

16. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ **x - 1 = racine(x);**
- ☐ **x = racine(2/3);**
- ☐ **x = racine(x \* x) - racine(x);**
- ☐ **x = racine(racine(x)\*racine(x));**

17. Au début de la fonction **main()** on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ B
- ☐ b
- ☐ A
- ☐ C

18. Pour afficher à l'aide de **printf("%d\n",tab[i]);** le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ **for(i=0;i<5;i=i+1)**
- ☐ **for(i=0;i<=5;i=i+1)**
- ☐ **for(i=1;i<5;i=i+1)**
- ☐ **for(i=1;i<=5;i=i+1)**

19. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 5
- ☐ 1

20. Si **pgcd** est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ **n = pgcd(n, 3);**
- ☐ **int pgcd(2);**
- ☐ **n = pgcd(int p, int q);**
- ☐ **int n = pgcd();**



**Barème : 1 points par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101  
☐ 3  
☐ 5  
☐ 4

2. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`  
☐ `int saisie_utilisateur();`  
☐ `void saisie_utilisateur(int n);`  
☐ `void saisie_utilisateur(char c);`

3. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`  
☐ `n = pgcd(int p, int q);`  
☐ `int pgcd(2);`  
☐ `int n = pgcd();`

4. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int n = carre();`  
☐ `int carre(2);`  
☐ `n = carre(int n);`  
☐ `n = carre(n);`

5. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
}
```

```
}
printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\*  
☐ \*\* \*\*\*  
☐ \*\*\*\*  
☐ \*\* \*\*

6. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ `!(A || B) == (A && !B)`  
☐ `(!A || B)`  
☐ `A && B`  
☐ `(A == TRUE) && (B == TRUE)`

7. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ des processus  
☐ certaines données de la mémoire de travail  
☐ en temps d'accès  
☐ les fichiers du disque

8. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, int x);`  
☐ `int pgcd(int y, int x);`  
☐ `int pgcd(int x, y);`

9. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x y);`  
☐ `printf("x=%x et y=%y\n");`  
☐ `printf("x=%d et y=%d\n,x,y");`  
☐ `printf("x=%d et y=%d\n",x,y);`

10. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui ne sera pas détectée  
☐ ne comportent aucune erreur  
☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique  
☐ comportent une erreur qui sera détectée au cours de l'édition de lien

11. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1  
☐ 4  
☐ 5  
☐ 0

12. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3  
☐ 7  
☐ 8  
☐ 111

13. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade en trop
- ☐ un point-virgule manquant
- ☐ un point-virgule en trop
- ☐ une accolade manquante

14. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 4
- ☐ 0
- ☐ 16

15. Sur unix (ou linux), la commande **mkdir** permet de :

- ☐ créer un fichier texte
- ☐ ouvrir un fichier texte

- ☐ créer un répertoire
- ☐ changer de répertoire courant

16. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie

17. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char 'c';`
- ☐ `char c;`

18. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `begin`
- ☐ `init`
- ☐ `main`
- ☐ `include`

19. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ le bus explose
- ☐ la case mémoire 8 contiendra 16
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0

20. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(double n);`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42
- ☐ 0
- ☐ 6
- ☐ 1

2. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int pgcd(2);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `int n = pgcd();`
- ☐ `n = pgcd(n, 3);`

3. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 8
- ☐ 4
- ☐ 0

4. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 3

5. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int y, int x);`

6. Un fichier source est :

- ☐ un document de référence du système
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document illisible pour les humains
- ☐ un document qui doit être protégé

7. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration

8. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(int n);`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `saisie_utilisateur scanf(%d);`
- ☐ `int saisie_utilisateur();`

9. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade en trop
- ☐ une accolade manquante
- ☐ un point-virgule manquant
- ☐ un point-virgule en trop

10. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#include <studio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studlib.h>`

11. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b
- ☐ B
- ☐ A
- ☐ C

12. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur
- ☐ que l'on veut voir tous les avertissements

13. Vous utilisez une boucle **while** quand :

- ☐ vous avez déjà fait un **for** dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

14. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8
- ☐ 3
- ☐ 111
- ☐ 7

15. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction **f** ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(3,5)=8, a=3, b=5

- ☐ f(a,b)=8, a=3, b=5

- ☐ f(a,b)=8, a=8, b=5

- ☐ f(a,b)=13, a=8, b=5

16. Soit la fonction **g** définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression **g(0)** prendra la valeur :

- ☐ 7
- ☐ 5
- ☐ 0

17. Sous unix (ou linux), la commande **cd** permet de :

- ☐ ouvrir un bureau partagé (common desktop)
- ☐ détruire un fichier
- ☐ récupérer un programme arrêté avec la commande **ab**
- ☐ jouer de la musique
- ☐ changer de répertoire courant

18. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique

- ☐ comportent une erreur qui ne sera pas détectée

- ☐ ne comportent aucune erreur

- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien

19. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 4
- ☐ 5
- ☐ 1
- ☐ 0

20. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ main
- ☐ init
- ☐ begin
- ☐ include

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1  
☐ 5  
☐ 0  
☐ 4

2. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien  
☐ Majeur  
☐ Mineur  
☐ Mineur  
Majeur

3. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction

- ☐ dans lequel ces fonctions sont appelées dans le `main`  
☐ un ordre quelconque  
☐ alphabétique

4. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`  
☐ `f(3,5)=8, a=3, b=5`  
☐ `f(a,b)=8, a=3, b=5`  
☐ `f(a,b)=13, a=8, b=5`

5. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max' ,** que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration  
☐ une directive préprocesseur `#include` manquante  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction déclarée mais non définie

6. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`  
☐ `int n = carre();`  
☐ `n = carre(int n);`  
☐ `n = carre(n);`

7. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur  
Majeur  
☐ Mineur  
☐ rien  
☐ Majeur

8. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8  
☐ 7  
☐ 3  
☐ 111

9. Sous unix (ou linux), la commande `cd` permet de :

- ☐ détruire un fichier  
☐ ouvrir un bureau partagé (common desktop)  
☐ récupérer un programme arrêté avec la commande `ab`  
☐ changer de répertoire courant  
☐ jouer de la musique

10. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(2/3);`  
☐ `x - 1 = racine(x);`  
☐ `x = racine(x * x) - racine(x);`  
☐ `x = racine(racine(x)*racine(x));`

11. Le langage C est un langage

- ☐ composé
- ☐ lu, écrit, parlé
- ☐ interprété
- ☐ compilé

12. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ Transférer des données et instructions entre processeur et mémoire
- ☐ transporter les processus du tourniquet au processeur
- ☐ Arriver à l'heure en cours

13. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int y, int x);`

14. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2 0
- ☐ 8 6 4 2
- ☐ 8 2
- ☐ 0 2 4 6 8

15. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ changer de répertoire courant
- ☐ créer un répertoire
- ☐ créer un fichier texte
- ☐ ouvrir un fichier texte

16. Un bit est :

- ☐ l'instruction qui met fin à un programme
- ☐ la longueur d'un mot mémoire
- ☐ un battement d'horloge processeur
- ☐ un chiffre binaire (0 ou 1)

17. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`  
, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration

18. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 5
- ☐ 4
- ☐ 3

19. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

- ☐ 0.5
- ☐ 0
- ☐ 1
- ☐ 1.5

20. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche `x`
- ☐ le programme affiche `****`
- ☐ la variable `x` vaut  $-\frac{1}{2}$
- ☐ la variable `x` vaut 16

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%x et y=%y\n");`  
☐ `printf("x=%d et y=%d\n",x,y);`  
☐ `printf("x=%d et y=%d\n,x,y");`  
☐ `printf("x=%d et y=%d\n",x y);`

2. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=13, a=8, b=5`  
☐ `f(3,5)=8, a=3, b=5`  
☐ `f(a,b)=8, a=3, b=5`

3. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu();`  
☐ `double afficher_menu();`  
☐ `char afficher_menu(printf("menu"));`  
☐ `void afficher_menu();`  
☐ `int afficher_menu(int char);`

4. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7  
☐ 8  
☐ 3  
☐ 111

5. Une variable booléenne est un variable :

- ☐ réelle positive  
☐ jamais nulle  
☐ NaN (not a number, qui n'est pas un nombre)  
☐ qui est vraie ou fausse  
☐ à laquelle une valeur vient d'être affectée

6. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 0  
☐ 1  
☐ 4  
☐ 5

7. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3  
☐ 0  
☐ 4

8. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée  
☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal  
☐ la division du programme en zones homogènes échoue  
☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur

9. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 4  
☐ 3  
☐ 5  
☐ 101

10. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante  
☐ une fonction appelée avant sa déclaration  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction déclarée mais non définie

11. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `mkdir TP4`  
☐ `new TP4`  
☐ `yppasswd`  
☐ `kwrite TP4`

12. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1
- ☐ 0 1 2 3
- ☐ 4 3 2 1 0

13. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int tab[] = 5;`
- ☐ `char tableau[5];`
- ☐ `int toto[taille=5];`
- ☐ `int toto[5];`
- ☐ `int[] new tableau(5);`

14. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ alphabétique

15. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur

16. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 0 1 2 3
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Majeur

18. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur scanf(%d);`

19. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ `!(A || B) == (A && !B)`
- ☐ `(!A || B)`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `A && B`

20. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5

2. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade manquante
- ☐ une accolade en trop
- ☐ un point-virgule en trop
- ☐ un point-virgule manquant

3. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ en temps d'accès
- ☐ les fichiers du disque
- ☐ certaines données de la mémoire de travail
- ☐ des processus

4. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

- ☐ printf("x=%d et y=%d\n,x,y");
- ☐ printf("x=%d et y=%d\n",x,y);
- ☐ printf("x=%x et y=%y\n");
- ☐ printf("x=%d et y=%d\n",x y);

5. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a{n, x} == b{n, x}
- ☐ a = b
- ☐ a == b

6. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ alphabétique

7. Si **carre** est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que n est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ int carre(2);
- ☐ n = carre(int n);
- ☐ n = carre(n);
- ☐ int n = carre();

8. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 1
- ☐ 5
- ☐ 0
- ☐ 4

9. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char c;
- ☐ int char;
- ☐ char "c";
- ☐ char 'c';

10. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42
- ☐ 1
- ☐ 0
- ☐ 6

11. Si x est une variable réelle (de type double) alors x = 3/2 lui affecte la valeur :

- ☐ 1.5
- ☐ 1
- ☐ 0
- ☐ 0.5

12. Vous utilisez une boucle `while` quand :
- ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

13. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
  - ☐ Majeur
  - ☐ Mineur
  - ☐ Mineur
  - ☐ Majeur
14. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
- ☐ `void pgcd(int x, int y);`
  - ☐ `int pgcd(int y, int x);`
  - ☐ `int pgcd(int x, y);`
  - ☐ `int pgcd(int x, int x);`

15. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7
- ☐ 111
- ☐ 8
- ☐ 3

16. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 0
- ☐ 4
- ☐ 8

17. L'ordonnancement par tourniquet permet :

- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'afficher des ronds colorés à l'écran
- ☐ de doubler la mémoire disponible
- ☐ d'entretenir l'illusion que les processus tournent en parallèle

18. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `struct toto_s toto;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `toto_s n, x;`
- ☐ `int toto.n = 3;`

19. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`

20. Sous unix (ou linux), la commande `cd` permet de :

- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ détruire un fichier
- ☐ changer de répertoire courant
- ☐ jouer de la musique
- ☐ ouvrir un bureau partagé (common desktop)

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur
- ☐ Écrire des données sur le disque dur
- ☐ Transférer des données et instructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours

2. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou

3. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max' , que doit-on chercher dans le programme ?**

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur `#include` manquante

4. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <studio.h>`
- ☐ `#include <stdio.h>`
- ☐ `#include <stdlib.h>`
- ☐ `#appart <stdlib.h>`

5. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1
- ☐ 4
- ☐ 0
- ☐ 5

6. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 0 1 2 3
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1

7. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction

8. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

9. Sous unix (ou linux), la commande `ls` permet de :

- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ compiler un programme
- ☐ voir des clips musicaux
- ☐ afficher le contenu d'un fichier texte

10. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 5
- ☐ 4
- ☐ 3
- ☐ 101

11. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a == b`
- ☐ `a{n, x} == b{n, x}`
- ☐ `a = b`

12. Quel est l'opérateur de différence en C :

- ☐ !=
- ☐ <>
- ☐ !
- ☐ ≠

13. Un bit est :

- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur

14. Si cette erreur apparaît à la compilation :

**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade en trop
- ☐ un point-virgule en trop
- ☐ une accolade manquante
- ☐ un point-virgule manquant

15. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle();`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle(double n);`

16. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction **f** ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`

17. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`

18. Sous unix (ou linux), la commande **cd** permet de :

- ☐ changer de répertoire courant
- ☐ détruire un fichier
- ☐ ouvrir un bureau partagé (common desktop)
- ☐ jouer de la musique
- ☐ récupérer un programme arrêté avec la commande **ab**

19. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tour à tour, un petit peu à chaque fois
- ☐ en parallèle, chacun dans un registre
- ☐ tous ensemble
- ☐ chacun son tour, après que le processus précédent a terminé

20. Si cet avertissement apparaît à la compilation :

**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur **#include** manquante
- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur
- ☐ Arriver à l'heure en cours

2. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 5
- ☐ 3
- ☐ 4
- ☐ 101

3. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`
- ☐ `int factorielle(double n);`

4. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 4
- ☐ 5
- ☐ 1
- ☐ 0

5. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 1 2 3 4
- ☐ 4 3 2 1

6. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 0
- ☐ 8
- ☐ 16
- ☐ 4

7. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 0 1 2 3

8. Le langage C est un langage

- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ interprété
- ☐ composé

9. Soit la fonction **g** définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression **g(0)** prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

10. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 16
- ☐ le bus explose
- ☐ la case mémoire 8 contiendra 0
- ☐ le terminal affiche 8

11. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur **#include** manquante
- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration

12. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`

13. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

14. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui ne sera pas détectée
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien

15. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `struct toto_s toto;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `toto_s n, x;`
- ☐ `int toto.n = 3;`

16. Si cet avertissement apparaît à la compilation :

`warning: implicit declaration of function 'max', que doit-on chercher dans le programme ?`

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction appelée avant sa déclaration
- ☐ une fonction déclarée mais non définie

17. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

18. Un registre du processeur est :

- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une gamme de fréquence de fonctionnement du processeur

19. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ retourner un bloc
- ☐ répéter un bloc tant qu'une condition est vérifiée
- ☐ mettre les blocs en séquence les uns à la suite des autres

20. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un `for` dans le même programme principal

**Barème : 1 point par réponse juste (unique); −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2
- ☐ 8 6 4 2 0
- ☐ 0 2 4 6 8
- ☐ 8 2

2. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`

3. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 1 2 3 4

4. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy(p, q);`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy();`

5. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

6. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 4 3 2 1

7. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`

8. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`

9. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`
- ☐ `a == b`
- ☐ `(a.n == b.n) && (a.x == b.x)`

10. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou

11. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<n) || (n>b)`

- |  |   |   |
|--|---|---|
| <p>12. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :</p> <ul style="list-style-type: none"><li><input type="checkbox"/> <code>kwrite</code> TP4</li><li><input type="checkbox"/> <code>mkdir</code> TP4</li><li><input type="checkbox"/> <code>new</code> TP4</li><li><input type="checkbox"/> <code>yppasswd</code></li></ul> <p>13. Pour déclarer une procédure <code>afficher_menu</code> sans argument et qui ne renvoie rien on utilise :</p> <ul style="list-style-type: none"><li><input type="checkbox"/> <code>double afficher_menu();</code></li><li><input type="checkbox"/> <code>int afficher_menu();</code></li><li><input type="checkbox"/> <code>void afficher_menu();</code></li><li><input type="checkbox"/> <code>int afficher_menu(int char);</code></li><li><input type="checkbox"/> <code>char afficher_menu(sprintf("menu"));</code></li></ul> <p>14. Si <math>n</math> est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :</p> <ul style="list-style-type: none"><li><input type="checkbox"/> un débogueur</li><li><input type="checkbox"/> <code>printf("Valeur de n ? %g\n", n);</code></li><li><input type="checkbox"/> <code>printf("Valeur de n ? %d\n", n);</code></li><li><input type="checkbox"/> <code>scanf("%d", &amp;n);</code></li></ul> | <p>15. Si cet avertissement apparaît à la compilation :<br/><b>warning: implicit declaration of function 'max'</b>, que doit-on chercher dans le programme ?</p> <ul style="list-style-type: none"><li><input type="checkbox"/> une fonction appelée avant sa déclaration</li><li><input type="checkbox"/> une fonction déclarée mais non définie</li><li><input type="checkbox"/> une directive préprocesseur <code>#include</code> manquante</li><li><input type="checkbox"/> un désaccord entre la déclaration et la définition d'une fonction</li></ul> <p>16. Quel est l'opérateur de différence en C :</p> <ul style="list-style-type: none"><li><input type="checkbox"/> <code>!=</code></li><li><input type="checkbox"/> <code>≠</code></li><li><input type="checkbox"/> <code>!</code></li><li><input type="checkbox"/> <code>&lt;&gt;</code></li></ul> <p>17. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :</p> <ul style="list-style-type: none"><li><input type="checkbox"/> alphabétique</li><li><input type="checkbox"/> un ordre quelconque</li><li><input type="checkbox"/> dans lequel vous avez déclaré ces fonction</li><li><input type="checkbox"/> dans lequel ces fonctions sont appelées dans le <code>main</code></li></ul> | <p>18. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :<br/><b>Undefined symbols : "_printf" ou référence indéfinie vers &lt; printf &gt;</b></p> <ul style="list-style-type: none"><li><input type="checkbox"/> l'édition de liens</li><li><input type="checkbox"/> l'analyse des entrées clavier</li><li><input type="checkbox"/> l'analyse harmonique</li><li><input type="checkbox"/> l'analyse sémantique</li></ul> <p>19. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?</p> <ul style="list-style-type: none"><li><input type="checkbox"/> <code>char c;</code></li><li><input type="checkbox"/> <code>char "c";</code></li><li><input type="checkbox"/> <code>int char;</code></li><li><input type="checkbox"/> <code>char 'c';</code></li></ul> <p>20. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :</p> <ul style="list-style-type: none"><li><input type="checkbox"/> certaines données de la mémoire de travail</li><li><input type="checkbox"/> en temps d'accès</li><li><input type="checkbox"/> les fichiers du disque</li><li><input type="checkbox"/> des processus</li></ul> |
|--|---|---|



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2  
☐ 0 1 0 1 0 1 0 1  
☐ 1 2 1 2 3  
☐ 0 0 0 1 1 1

2. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`  
☐ `char c;`  
☐ `char "c";`  
☐ `int char;`

3. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`  
☐ `exposant(double x, int n, int r);`  
☐ `int exposant(double n, int x);`  
☐ `double exposant(double x, int n);`

4. Si  $x$  est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1  
☐ 1.5  
☐ 0  
☐ 0.5

5. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`  
☐ `#appart <stdlib.h>`  
☐ `#include <studio.h>`  
☐ `#include <stdio.h>`

6. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A  
☐ b  
☐ B  
☐ C

7. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs  
☐ ses cases  
☐ ses blocs  
☐ ses chants

8. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`  
☐ `(a<=n) && (n<=b)`  
☐ `(n<=a) && (n<=b)`  
☐ `(a<n) || (n>b)`

9. Si cette erreur apparaît à la compilation : `error: expected ';' before '}' token` que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant  
☐ une accolade en trop  
☐ un point-virgule en trop  
☐ une accolade manquante

10. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `int afficher_date(date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`  
☐ `void afficher_date(date_s d);`  
☐ `void afficher_date(struct date_s d);`

11. Si cette erreur apparaît à la compilation : `erreur: conflicting types for 'max'`, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie  
☐ une directive préprocesseur `#include` manquante  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction appelée avant sa déclaration

12. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0  
☐ 7  
☐ 5

13. Un fichier source est :

- ☐ un document illisible pour les humains
- ☐ un document qui doit être protégé
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document de référence du système
- ☐ un fichier texte qui sera traduit en instructions processeur

14. Le code suivant :

```
int i;  
for (i = 0; i < 5; i = i + 1)  
{  
    printf("%d ", i);  
}  
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 0 1 2 3
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4

15. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int[] nouveau(5);`
- ☐ `int tab[] = 5;`
- ☐ `char tableau[5];`
- ☐ `int toto[5];`
- ☐ `int toto[taille=5];`

16. Soit un programme contenant les lignes suivantes :

```
int i = 0;  
int j = 0;  
for (i = 0; i < 3; i = i + 1)  
{  
    for (j = 0; j < 5; j = j + 1)  
    {  
        ...  
    }  
}  
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 5
- ☐ j = %d
- ☐ j = 4
- ☐ j = 0

17. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur
- ☐ que l'on veut voir tous les avertissements

18. Une variable booléenne est un variable :

- ☐ qui est vraie ou fausse
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ à laquelle une valeur vient d'être affectée
- ☐ jamais nulle
- ☐ réelle positive

19. Pour l'extrait de programme suivant :

```
int i = 0;  
int j = 0;  
for (i = 0; i < 3; i = i + 1)  
{  
    for (j = 0; j < 2; j = j + 1)  
    {  
        printf("%d ", i);  
    }  
}  
printf("\n");
```

qu'est ce qui sera affiché?

- ☐ 0 1 0 1 0 1
- ☐ 0 0 1 1 2 2
- ☐ 0 1 2 0 1 2
- ☐ 1 2 3 1 2

20. Le code suivant :

```
int i;  
for (i = 8; i > 0; i = i - 2)  
{  
    printf("%d ", i);  
}  
printf("\n");
```

affichera :

- ☐ 8 6 4 2 0
- ☐ 8 2
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ include  
☐ main  
☐ init  
☐ begin

2. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`  
☐ `int pgcd(2);`  
☐ `n = pgcd(n, 3);`  
☐ `n = pgcd(int p, int q);`

3. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6  
☐ 0  
☐ 10  
☐ 15

4. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs  
☐ ses cases  
☐ ses blocs  
☐ ses chants

5. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x - 1 = racine(x);`  
☐ `x = racine(2/3);`  
☐ `x = racine(racine(x)*racine(x));`  
☐ `x = racine(x * x) - racine(x);`

6. Le langage C est un langage

- ☐ composé  
☐ lu, écrit, parlé  
☐ interprété  
☐ compilé

7. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un fichier texte  
☐ changer de répertoire courant  
☐ créer un répertoire  
☐ ouvrir un fichier texte

8. On considère deux variables booléennes A et B initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(A == TRUE) && (B == TRUE)`  
☐ `!(A || B) == (A && !B)`  
☐ `A && B`  
☐ `(!A || B)`

9. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`  
☐ `int afficher_date(date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`  
☐ `void afficher_date(struct date_s d);`

10. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique  
☐ analyse syntaxique  
☐ analyse sémantique  
☐ analyse lexicale

11. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas  
☐ il n'affiche rien  
☐ il comporte une boucle infinie  
☐ il risque d'afficher bonjour à la place de coucou

12. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`  
☐ `printf("x=%d et y=%d\n",x y);`  
☐ `printf("x=%d et y=%d\n,x,y");`  
☐ `printf("x=%x et y=%y\n");`

13. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur  
☐ Mineur  
☐ rien  
☐ Mineur  
Majeur

14. Le type des réels en C est :

- ☐ `double`  
☐ `char`  
☐ `int`  
☐ `real`

15. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine

16. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
}
```

```
printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\*
- ☐ \*\* \*\*\*
- ☐ \*\*\*\*\*
- ☐ \*\*\*\*
- ☐ \*\* \*\*

17. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu();`
- ☐ `double afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `void afficher_menu();`
- ☐ `char afficher_menu(printf("menu"));`

18. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 111
- ☐ 7
- ☐ 3
- ☐ 8

19. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `char c;`

20. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 8 6 4 2
- ☐ 8 6 4 2 0
- ☐ 8 2

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
  - ☐ `for(i=0; i<=5; i=i+1)`
  - ☐ `for(i=1; i<=5; i=i+1)`
  - ☐ `for(i=1; i<5; i=i+1)`
  - ☐ `for(i=0; i<5; i=i+1)`
- Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
  - ☐ `int exposant(double n, int x);`
  - ☐ `double exposant(double x, int n);`
  - ☐ `exposant(double x, int n, int r);`
  - ☐ `void exposant(double x^n);`
- Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction déclarée mais non définie
  - ☐ une fonction appelée avant sa déclaration
  - ☐ une directive préprocesseur `#include` manquante
- Après exécution du programme :
 

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

  - ☐ la case mémoire 8 contiendra 16
  - ☐ la case mémoire 8 contiendra 0
  - ☐ le bus explose
  - ☐ le terminal affiche 8

- Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
  - ☐ `int pgcd(int y, int x);`
  - ☐ `void pgcd(int x, int y);`
  - ☐ `int pgcd(int x, int x);`
  - ☐ `int pgcd(int x, y);`
- Soient deux variables entières  $x$  et  $y$  initialisées à 4 et 5 respectivement. L'affichage  $x=4$  et  $y=5$  est obtenu avec la commande :
  - ☐ `printf("x=%x et y=%y\n");`
  - ☐ `printf("x=%d et y=%d\n", x y);`
  - ☐ `printf("x=%d et y=%d\n", x, y);`
  - ☐ `printf("x=%d et y=%d\n, x, y");`
- Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :
  - ☐ `n = pgcd(n, 3);`
  - ☐ `int n = pgcd();`
  - ☐ `n = pgcd(int p, int q);`
  - ☐ `int pgcd(2);`
- Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :
  - ☐ `yppasswd`
  - ☐ `kwrite TP4`
  - ☐ `mkdir TP4`
  - ☐ `new TP4`
- Si  $a$  et  $b$  sont deux variables de type :
 

```
struct toto_s
{
    int n;
    double x;
};
```

 Alors pour tester l'égalité de  $a$  et de  $b$  on utilise la condition :
  - ☐ `a{n, x} == b{n, x}`
  - ☐ `a == b`
  - ☐ `(a.n == b.n) && (a.x == b.x)`
  - ☐ `a = b`

- Sous unix (ou linux), la commande `ls` permet de :
  - ☐ voir des clips musicaux
  - ☐ afficher la liste de fichiers contenus dans un répertoire
  - ☐ afficher le contenu d'un fichier texte
  - ☐ compiler un programme
- Soit le programme principal suivant :
 

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n", f(a,b), a, b);
    return EXIT_SUCCESS;
}
```

 appelant la fonction `f` ainsi définie :
 

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

 L'affichage dans le main est le suivant :
  - ☐ `f(a,b)=8, a=3, b=5`
  - ☐ `f(a,b)=13, a=8, b=5`
  - ☐ `f(3,5)=8, a=3, b=5`
  - ☐ `f(a,b)=8, a=8, b=5`
- Un fichier source est :
  - ☐ un document illisible pour les humains
  - ☐ un document de référence du système
  - ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
  - ☐ un fichier texte qui sera traduit en instructions processeur
  - ☐ un document qui doit être protégé
- Vous utilisez une boucle `while` quand :
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ l'incrément de la variable de boucle n'est pas 1

14. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
- ☐ `printf("%d", factorielle(n));`
  - ☐ `n = factorielle();`
  - ☐ `int factorielle(int 2);`
  - ☐ `n = factorielle(p, q);`
15. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :
- ☐ `saisie_utilisateur(scanf(%d));`
  - ☐ `void saisie_utilisateur(int n);`
  - ☐ `int saisie_utilisateur();`
  - ☐ `void saisie_utilisateur(char c);`
16. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ la variable `x` vaut  $-\frac{1}{2}$
- ☐ le programme affiche `****`
- ☐ la variable `x` vaut 16
- ☐ le programme affiche `x`

17. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ `**** * * * * *`
- ☐ `** *** * * * * *`
- ☐ `** ** * * * * *`
- ☐ `***** * * * *`

18. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses blocs
  - ☐ ses chants
  - ☐ ses cases
  - ☐ ses champs
19. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :
- ☐ retourner un bloc
  - ☐ répéter un bloc tant qu'une condition est vérifiée
  - ☐ sélectionner entre deux blocs à l'aide d'une condition
  - ☐ mettre les blocs en séquence les uns à la suite des autres
20. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
- ☐ `#include <stdlib.h>`
  - ☐ `#include <studio.h>`
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <stdio.h>`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4  
☐ 0  
☐ 3

2. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(!A || B)`  
☐ `A && B`  
☐ `(A == TRUE) && (B == TRUE)`  
☐ `!(A || B) == (A && !B)`

3. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases  
☐ ses blocs  
☐ ses chants  
☐ ses champs

4. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 1.5  
☐ 1  
☐ 0  
☐ 0.5

5. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `init`  
☐ `include`  
☐ `main`  
☐ `begin`

6. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source  
☐ que l'on veut voir tous les avertissements  
☐ qu'il faut lancer un débogueur  
☐ qu'on veut changer aléatoirement de fond d'écran

7. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int n = carre();`  
☐ `int carre(2);`  
☐ `n = carre(n);`  
☐ `n = carre(int n);`

8. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0  
☐ 4 3 2 1  
☐ 0 1 2 3  
☐ 0 1 2 3 4

9. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.exe -o prog.c`  
☐ `gcc prog.exe -Wall -o prog.c`  
☐ `gcc -Wall prog.c -o prog.exe`  
☐ `gcc prog.c -o -Wall prog.exe`

10. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int toto.n = 3;`  
☐ `toto_s n, x;`  
☐ `int struct toto_s = {3, -1e10};`  
☐ `struct toto_s toto;`  
☐ `toto_s struct z = {3, 0.5};`

11. Sous unix (ou linux), pour créer un répertoire `TP4` dans le répertoire courant on peut utiliser la commande :

- ☐ `mkdir TP4`  
☐ `yppasswd`  
☐ `new TP4`  
☐ `kwrite TP4`

12. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `int afficher_date(date_s d);`  
☐ `void afficher_date(struct date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`  
☐ `void afficher_date(date_s d);`

13. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle(p, q);`  
☐ `n = factorielle();`  
☐ `printf("%d", factorielle(n));`  
☐ `int factorielle(int 2);`

14. Le bus système sert à :

- ☐ Arriver à l'heure en cours  
☐ transporter les processus du tourniquet au processeur  
☐ Écrire des données sur le disque dur  
☐ Transférer des données et intructions entre processeur et mémoire

15. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

☐ `int toto[taille=5];`  
☐ `char tableau[5];`  
☐ `int toto[5];`  
☐ `int tab[] = 5;`  
☐ `int[] new tableau(5);`

16. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
```

`printf("j = %d\n", j);`

qu'est ce qui sera affiché par ce printf?

☐ `j = 0`

☐ `j = 4`

☐ `j = 5`

☐ `j = %d`

17. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

☐ `double exposant(double x, int n);`  
☐ `void exposant(double x^n);`  
☐ `int exposant(double n, int x);`  
☐ `exposant(double x, int n, int r);`

18. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
```

`printf("produit = %d", produit);`

La valeur affichée est :

☐ 16

☐ 0

☐ 4

☐ 8

19. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

☐ une accolade en trop  
☐ un point-virgule en trop  
☐ un point-virgule manquant  
☐ une accolade manquante

20. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

☐ il ne compile pas  
☐ il risque d'afficher bonjour à la place de coucou  
☐ il comporte une boucle infinie  
☐ il n'affiche rien



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 7  
☐ 0  
☐ 5

2. Après exécution du programme :

```
1 lecture 8 r0
2 valeur 3 r1
3 mult r1 r0
4 valeur 1 r2
5 add r2 r0
6 ecriture r0 8
7 stop
8 5
```

- ☐ le bus explose  
☐ la case mémoire 8 contiendra 16  
☐ la case mémoire 8 contiendra 0  
☐ le terminal affiche 8

3. Le bus système sert à :

- ☐ Écrire des données sur le disque dur  
☐ transporter les processus du tourniquet au processeur  
☐ Arriver à l'heure en cours  
☐ Transférer des données et intructions entre processeur et mémoire

4. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction main(), alors on peut écrire en début de main() :

- ☐ toto\_s n, x;  
☐ int toto.n = 3;  
☐ struct toto\_s toto;  
☐ toto\_s struct z = {3, 0.5};  
☐ int struct toto\_s = {3, -1e10};

5. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 4  
☐ 8  
☐ 0  
☐ 16

6. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3  
☐ 16  
☐ 20  
☐ 6

7. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie  
☐ il ne compile pas  
☐ il risque d'afficher bonjour à la place de coucou  
☐ il n'affiche rien

8. Le type des réels en C est :

- ☐ char  
☐ int  
☐ real  
☐ double

9. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 10  
☐ 0  
☐ 15  
☐ 6

10. Une variable booléenne est une variable :

- ☐ jamais nulle  
☐ NaN (not a number, qui n'est pas un nombre)  
☐ réelle positive  
☐ qui est vraie ou fausse  
☐ à laquelle une valeur vient d'être affectée

11. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui ne sera pas détectée
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique

12. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ en parallèle, chacun dans un registre
- ☐ tour à tour, un petit peu à chaque fois
- ☐ tous ensemble

13. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 1
- ☐ 4

14. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `int saisie_utilisateur();`

15. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Mineur Majeur
- ☐ Majeur
- ☐ rien

16. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ changer de répertoire courant
- ☐ créer un fichier texte
- ☐ créer un répertoire
- ☐ ouvrir un fichier texte

17. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `char afficher_menu(printf("menu"));`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `double afficher_menu();`
- ☐ `int afficher_menu();`

18. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

19. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5

20. Un fichier source est :

- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document qui doit être protégé
- ☐ un document de référence du système
- ☐ un document illisible pour les humains
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0  
☐ 5  
☐ 7

2. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ cccccc  
☐ ABCDEF  
☐ A  
☐ i

3. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a == b`  
☐ `(a.n == b.n) && (a.x == b.x)`  
☐ `a{n, x} == b{n, x}`  
☐ `a = b`

4. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`  
☐ `int pgcd(int x, int x);`  
☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, y);`

5. Quel est l'opérateur de différence en C :

- ☐ `!=`  
☐ `≠`  
☐ `!`  
☐ `<>`

6. Si cette erreur apparaît à la compilation :

**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante  
☐ un caractère interdit en C  
☐ une faute de frappe dans un appel de fonction  
☐ une variable non déclarée

7. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int[] new tableau(5);`  
☐ `int toto[5];`  
☐ `int tab[] = 5;`  
☐ `char tableau[5];`  
☐ `int toto[taille=5];`

8. Un bit est :

- ☐ la longueur d'un mot mémoire  
☐ un battement d'horloge processeur  
☐ un chiffre binaire (0 ou 1)  
☐ l'instruction qui met fin à un programme

9. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs  
☐ ses chants  
☐ ses cases  
☐ ses blocs

10. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1  
☐ 5  
☐ 0  
☐ 4

11. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `include`  
☐ `begin`  
☐ `main`  
☐ `init`

12. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ répéter un bloc tant qu'une condition est vérifiée  
☐ sélectionner entre deux blocs à l'aide d'une condition  
☐ mettre les blocs en séquence les uns à la suite des autres  
☐ retourner un bloc

13. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 0 0 1 1 1
- ☐ 1 2 1 2 3

14. Vous utilisez une boucle **while** quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un **for** dans le même programme principal

15. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
```

```
}
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = %d
- ☐ j = 5
- ☐ j = 0
- ☐ j = 4

16. Une variable booléenne est un variable :

- ☐ à laquelle une valeur vient d'être affectée
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ réelle positive
- ☐ jamais nulle
- ☐ qui est vraie ou fausse

17. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse syntaxique
- ☐ analyse sémantique
- ☐ analyse lexicale

18. Quels calculs peut-on programmer en programmation structurée ?

- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti

- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine

19. On souhaite faire une boucle de contrôle de saisie : tant que l'entier **n** n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de **n**. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition **cond** :

- ☐ (a<n) || (n>b)
- ☐ a<=n<=b
- ☐ (a<=n) && (n<=b)
- ☐ (n<=a) && (n<=b)

20. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ double exposant(double x, int n);
- ☐ exposant(double x, int n, int r);
- ☐ void exposant(double x^n);
- ☐ int exposant(double n, int x);

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<5;i=i+1)`  
☐ `for(i=1;i<5;i=i+1)`  
☐ `for(i=0;i<=5;i=i+1)`  
☐ `for(i=1;i<=5;i=i+1)`

2. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `init`  
☐ `main`  
☐ `begin`  
☐ `include`

3. Après exécution jusqu'à la ligne 15 du programme C :

```

10  ...
11  int main() {
12      int x = 5;
13
14      x = 3 * x + 1;
15
16      ...
17  }
```

- ☐ le programme affiche \*\*\*\*  
☐ le programme affiche x  
☐ la variable x vaut  $-\frac{1}{2}$   
☐ la variable x vaut 16

4. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse sémantique  
☐ analyse harmonique  
☐ analyse syntaxique  
☐ analyse lexicale

5. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants  
☐ ses blocs  
☐ ses cases  
☐ ses champs

6. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ `loop i;`  
☐ `int k;`  
☐ `int loop n;`  
☐ `int %d;`

7. Soit la fonction f définie par :

```

int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4  
☐ 5  
☐ 0  
☐ 1

8. On considère deux variables booléennes A et B initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(A == TRUE) && (B == TRUE)`  
☐ `A && B`  
☐ `(!A || B)`  
☐ `!(A || B) == (A && !B)`

9. Une variable booléenne est un variable :

- ☐ jamais nulle  
☐ réelle positive  
☐ à laquelle une valeur vient d'être affectée  
☐ NaN (not a number, qui n'est pas un nombre)  
☐ qui est vraie ou fausse

10. Après exécution jusqu'à la ligne 15 du programme C :

```

10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

- ☐ la variable x vaut 3  
☐ le programme affiche "Faux"  
☐ la variable x vaut 5 et la variable y vaut 3  
☐ la variable y vaut 5

11. Pour l'extrait de programme suivant :

```

int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16  
☐ 0  
☐ 8  
☐ 4

12. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que n est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`  
☐ `n = carre(n);`  
☐ `n = carre(int n);`  
☐ `int n = carre();`

13. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`

14. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`

15. Le langage C est un langage

- ☐ interprété
- ☐ lu, écrit, parlé
- ☐ composé
- ☐ compilé

16. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`  
, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration

- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie

17. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 6
- ☐ 42
- ☐ 0
- ☐ 1

18. Si cette erreur apparaît à la compilation :

`error: expected ';' before '}' token` que doit-on chercher dans le programme ?

- ☐ une accolade en trop
- ☐ un point-virgule manquant

- ☐ un point-virgule en trop
- ☐ une accolade manquante

19. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `struct toto_s toto;`
- ☐ `int toto.n = 3;`
- ☐ `toto_s n, x;`

20. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ les fichiers du disque
- ☐ des processus
- ☐ en temps d'accès
- ☐ certaines données de la mémoire de travail

2. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ un débogueur

3. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2
- ☐ 1 2 3 1 2
- ☐ 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2

4. Soient deux variables entières  $x$  et  $y$  initialisées à 4 et 5 respectivement. L'affichage  $x=4$  et  $y=5$  est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x,y);`

5. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `void afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu();`
- ☐ `double afficher_menu();`

6. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction  $f$  ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=3, b=5`

7. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur
- ☐ Mineur Majeur
- ☐ Majeur
- ☐ rien

8. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 0 1 2 3
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4

9. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur Majeur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

10. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`
- ☐ `x = mccarthy(n);`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy(p, q);`

11. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `int char;`

12. Après exécution jusqu'à la ligne 15 du programme C :

```
10  ...
11  int main() {
12      int x = 5;
13
14      x = 3 * x + 1;
15
16      ...
17  }
```

- ☐ la variable x vaut 16
- ☐ le programme affiche \*\*\*\*
- ☐ le programme affiche x
- ☐ la variable x vaut  $-\frac{1}{2}$

13. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal

14. Si cet avertissement apparaît à la compilation :

`warning: implicit declaration of function 'max', que doit-on chercher dans le programme ?`

- ☐ une directive préprocesseur `#include` manquante

☐ un désaccord entre la déclaration et la définition d'une fonction

- ☐ une fonction appelée avant sa déclaration
- ☐ une fonction déclarée mais non définie

15. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(2/3);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(x * x) - racine(x);`

16. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `struct toto_s toto;`
- ☐ `toto_s n, x;`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `int toto.n = 3;`

17. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle();`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(double n);`
- ☐ `int factorielle(int x);`

18. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

19. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int y, int x);`

20. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur



**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ (A == TRUE) && (B == TRUE)
- ☐ A && B
- ☐ (!A || B)
- ☐ !(A || B) == (A && !B)

2. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 0 1 2 3 4

3. Le type des réels en C est :

- ☐ char
- ☐ double
- ☐ real
- ☐ int

4. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `saisie_utilisateur scanf(%d);`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`

5. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses champs
- ☐ ses chants
- ☐ ses blocs

6. Un fichier source est :

- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document illisible pour les humains
- ☐ un document qui doit être protégé
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document de référence du système

7. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ cccccc
- ☐ A
- ☐ ABCDEF
- ☐ i

8. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien
- ☐ il ne compile pas

9. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = %d
- ☐ j = 5
- ☐ j = 4
- ☐ j = 0

10. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %d\n", n);`

11. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 4

☐ j = %d

☐ j = 0

☐ j = 5

12. Un bit est :

☐ un battement d'horloge processeur

☐ la longueur d'un mot mémoire

☐ l'instruction qui met fin à un programme

☐ un chiffre binaire (0 ou 1)

13. Avant de faire appel à une fonction il est nécessaire de :

☐ l'avoir déclarée

☐ avoir défini une constante symbolique de la taille de cette fonction

☐ l'avoir définie

☐ l'avoir déclarée et définie

14. Au début de la fonction `main()` on place le code :

```
char b = 'A';  
b = b + 2;  
printf("%c\n", b);
```

Alors l'affichage sera :

☐ A

☐ B

☐ b

☐ C

15. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

☐ `for(i=1;i<5;i=i+1)`

☐ `for(i=0;i<=5;i=i+1)`

☐ `for(i=0;i<5;i=i+1)`

☐ `for(i=1;i<=5;i=i+1)`

16. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

☐ `int n = carre();`

☐ `n = carre(int n);`

☐ `int carre(2);`

☐ `n = carre(n);`

17. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

☐ une fonction déclarée mais non définie

☐ une fonction appelée avant sa déclaration

☐ une directive préprocesseur `#include` manquante

☐ un désaccord entre la déclaration et la définition d'une fonction

18. Le code suivant :

```
int age = 20;  
if (age < 18)  
{  
    printf("Mineur\n");  
}  
printf("Majeur\n");
```

affichera :

☐ Majeur

☐ Mineur

☐ Mineur

Majeur

☐ rien

19. Le code suivant :

```
int somme = 0;  
int i;  
for (i = 1; i < 4; i = i + 1)  
{  
    somme = somme + i;  
}  
printf("%d", somme);
```

affichera :

☐ 42

☐ 1

☐ 6

☐ 0

20. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

☐ `printf("x=%d et y=%d\n",x y);`

☐ `printf("x=%x et y=%y\n");`

☐ `printf("x=%d et y=%d\n,x,y");`

☐ `printf("x=%d et y=%d\n",x,y);`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1

2. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque

3. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses cases
- ☐ ses champs
- ☐ ses chants

4. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée
- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir définie
- ☐ l'avoir déclarée et définie

5. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=8, b=5

6. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 5
- ☐ j = %d
- ☐ j = 0
- ☐ j = 4

7. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui ne sera pas détectée

8. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché?

- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 3 0 1 2

9. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche x = 2
- ☐ le terminal affiche "Faux"
- ☐ le terminal affiche 5
- ☐ le terminal affiche x = 5

10. Une variable booléenne est un variable :

- ☐ qui est vraie ou fausse
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ réelle positive
- ☐ jamais nulle
- ☐ à laquelle une valeur vient d'être affectée

11. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut lancer un débogueur
- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements

12. L'ordonnancement par tourniquet permet :

- ☐ de doubler la mémoire disponible
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ d'afficher des ronds colorés à l'écran

13. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`
- ☐ `int pgcd(2);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `n = pgcd(n, 3);`

14. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `void afficher_menu();`
- ☐ `double afficher_menu();`
- ☐ `char afficher_menu(sprintf("menu"));`
- ☐ `int afficher_menu(int char);`
- ☐ `int afficher_menu();`

15. Sous unix (ou linux), la commande `ls` permet de :

- ☐ compiler un programme
- ☐ voir des clips musicaux
- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ afficher le contenu d'un fichier texte

16. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `char "c";`
- ☐ `char c;`
- ☐ `int char;`
- ☐ `char 'c';`

17. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ ouvrir un fichier texte
- ☐ créer un répertoire
- ☐ créer un fichier texte
- ☐ changer de répertoire courant

18. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 1
- ☐ 5

19. Si cet avertissement apparaît à la compilation :

**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme?

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration

20. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ un débogueur
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `scanf("%d", &n);`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
  - ☐ une fonction appelée avant sa déclaration
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction déclarée mais non définie
2. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :
  - ☐ 42
  - ☐ 1
  - ☐ 6
  - ☐ 0
3. Pour compiler un programme `prog.c`, on utilise la ligne de commande :
  - ☐ `gcc prog.c -o -Wall prog.exe`
  - ☐ `gcc -Wall prog.exe -o prog.c`
  - ☐ `gcc prog.exe -Wall -o prog.c`
  - ☐ `gcc -Wall prog.c -o prog.exe`
4. Si cette erreur apparaît à la compilation :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?
  - ☐ un caractère interdit en C
  - ☐ une faute de frappe dans un appel de fonction
  - ☐ une variable non déclarée
  - ☐ une directive préprocesseur `#include` manquante

5. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :
  - ☐ `printf("x=%d et y=%d\n",x,y);`
  - ☐ `printf("x=%d et y=%d\n",x y);`
  - ☐ `printf("x=%x et y=%y\n");`
  - ☐ `printf("x=%d et y=%d\n",x,y);`
6. L'écriture 101 en binaire correspond au nombre naturel :
  - ☐ 5
  - ☐ 3
  - ☐ 101
  - ☐ 4
7. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :
  - ☐ rien
  - ☐ Mineur
  - ☐ Mineur Majeur
  - ☐ Majeur
8. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.
  - ☐ `#define N = 3`
  - ☐ `#define taille = N`
  - ☐ `#define taille = 3`
  - ☐ `#define N 3`
9. Une variable booléenne est une variable :
  - ☐ jamais nulle
  - ☐ réelle positive
  - ☐ à laquelle une valeur vient d'être affectée
  - ☐ NaN (not a number, qui n'est pas un nombre)
  - ☐ qui est vraie ou fausse

10. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :
  - ☐ 1
  - ☐ 0.5
  - ☐ 1.5
  - ☐ 0
11. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :
  - ☐ `char afficher_menu(printf("menu"));`
  - ☐ `void afficher_menu();`
  - ☐ `double afficher_menu();`
  - ☐ `int afficher_menu();`
  - ☐ `int afficher_menu(int char);`
12. Le bus système sert à :
  - ☐ transporter les processus du tourniquet au processeur
  - ☐ Arriver à l'heure en cours
  - ☐ Écrire des données sur le disque dur
  - ☐ Transférer des données et instructions entre processeur et mémoire
13. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

  - ☐ il comporte une boucle infinie
  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il ne compile pas
  - ☐ il n'affiche rien
14. Sous unix (ou linux), pour créer un répertoire `TP4` dans le répertoire courant on peut utiliser la commande :
  - ☐ `new TP4`
  - ☐ `mkdir TP4`
  - ☐ `yppasswd`
  - ☐ `kwrite TP4`

15. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy(p, q);`
- ☐ `n = mccarthy();`
- ☐ `x = mccarthy(n);`
- ☐ `int mccarthy(int 2);`

16. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `int factorielle(int 2);`
- ☐ `n = factorielle(p, q);`
- ☐ `printf("%d", factorielle(n));`
- ☐ `n = factorielle();`

17. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1

18. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `char c;`

19. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur

☐ rien

☐ Mineur  
Majeur

☐ Majeur

20. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

☐ 16

☐ 4

☐ 0

☐ 8

**Barème : 1 point par réponse juste (unique); –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ un ordre quelconque
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ dans lequel vous avez déclaré ces fonction

2. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `int exposant(double n, int x);`
- ☐ `void exposant(double x^n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `double exposant(double x, int n);`

3. Si  $a$  et  $b$  sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de  $a$  et de  $b$  on utilise la condition :

- ☐ `a == b`
- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`
- ☐ `(a.n == b.n) && (a.x == b.x)`

4. Un fichier source est :

- ☐ un document qui doit être protégé
- ☐ un document illisible pour les humains
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document de référence du système

5. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 0 1 1 1
- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
- ☐ 1 2 1 2 3

6. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses champs
- ☐ ses chants
- ☐ ses cases

7. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`

8. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle();`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(double n);`
- ☐ `int factorielle(int x);`

9. Si cette erreur apparaît à la compilation :

Undefined symbols : "\_printf" ou référence indéfinie vers < printf > que doit-on chercher dans le programme ?

- ☐ une faute de frappe dans un appel de fonction
- ☐ une variable non déclarée
- ☐ un caractère interdit en C
- ☐ une directive préprocesseur `#include` manquante

10. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1

11. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche `x = 2`
- ☐ le terminal affiche 5
- ☐ le terminal affiche "Faux"
- ☐ le terminal affiche `x = 5`

12. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `char "c";`

13. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`

14. Quel est l'opérateur de différence en C :

- ☐ `!`
- ☐ `!=`
- ☐ `<>`
- ☐ `≠`

15. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction appelée avant sa déclaration

16. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ la variable x vaut  $-\frac{1}{2}$
- ☐ la variable x vaut 16
- ☐ le programme affiche `****`
- ☐ le programme affiche x

17. Le type des réels en C est :

- ☐ `double`
- ☐ `int`
- ☐ `real`
- ☐ `char`

18. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
```

```
{
    printf("*");
}
printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ `**** **** **** ****`
- ☐ `** *** **** *****`
- ☐ `***** **** *** **`
- ☐ `** ** * * * * *`

19. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy(p, q);`
- ☐ `x = mccarthy(n);`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy();`

20. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 5
- ☐ 101
- ☐ 4



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 1 2 1 2 3  
☐ 0 1 2 0 1 2  
☐ 0 0 0 1 1 1  
☐ 0 1 0 1 0 1 0 1

2. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ A && B  
☐ (A == TRUE) && (B == TRUE)  
☐ !(A || B) == (A && !B)  
☐ (!A || B)

3. Pour déclarer une fonction pgcd qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ void pgcd(int x, int y);  
☐ int pgcd(int y, int x);  
☐ int pgcd(int x, y);  
☐ int pgcd(int x, int x);

4. Après exécution du programme :

```
1 lecture 8 r0
2 valeur 3 r1
3 mult r1 r0
4 valeur 1 r2
```

```
5 add r2 r0
6 ecriture r0 8
7 stop
8 5
```

- ☐ la case mémoire 8 contiendra 0  
☐ le terminal affiche 8  
☐ la case mémoire 8 contiendra 16  
☐ le bus explose

5. Au début de la fonction main() on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b  
☐ C  
☐ A  
☐ B

6. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas  
☐ il comporte une boucle infinie  
☐ il n'affiche rien  
☐ il risque d'afficher bonjour à la place de coucou

7. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 4  
☐ 8  
☐ 16  
☐ 0

8. Pour déclarer une procédure afficher\_date qui prend en argument un struct date\_s et affiche le contenu du struct, on écrit :

- ☐ void afficher\_date(struct date\_s d);  
☐ int afficher\_date(date\_s d);  
☐ void afficher\_date(date\_s d);  
☐ struct date\_s afficher\_date(struct date\_s d);

9. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(3,5)=8, a=3, b=5  
☐ f(a,b)=13, a=8, b=5  
☐ f(a,b)=8, a=3, b=5  
☐ f(a,b)=8, a=8, b=5

10. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ int[] new tableau(5);  
☐ char tableau[5];  
☐ int toto[taille=5];  
☐ int tab[] = 5;  
☐ int toto[5];

11. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou référence indéfinie vers « printf »`
- ☐ l'édition de liens
  - ☐ l'analyse sémantique
  - ☐ l'analyse des entrées clavier
  - ☐ l'analyse harmonique
12. Le code suivant :
- ```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 0 1 2 3
  - ☐ 0 1 2 3 4
  - ☐ 4 3 2 1 0
  - ☐ 4 3 2 1
13. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :
- ☐ `int pgcd(2);`
  - ☐ `n = pgcd(n, 3);`
  - ☐ `int n = pgcd();`
  - ☐ `n = pgcd(int p, int q);`
14. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ dans lequel vous avez déclaré ces fonction
  - ☐ alphabétique
  - ☐ dans lequel ces fonctions sont appelées dans le `main`
  - ☐ un ordre quelconque

15. Le code suivant :
- ```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```
- affichera :
- ☐ Mineur
  - ☐ Majeur
  - ☐ rien
  - ☐ Mineur
  - ☐ Majeur
16. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
- ☐ `for(i=1;i<5;i=i+1)`
  - ☐ `for(i=0;i<=5;i=i+1)`
  - ☐ `for(i=1;i<=5;i=i+1)`
  - ☐ `for(i=0;i<5;i=i+1)`
17. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses cases
  - ☐ ses blocs
  - ☐ ses champs
  - ☐ ses chants
18. Soit la fonction `f` définie par :
- ```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
}
```

- ```
}
return 4;
}
```
- Alors l'expression `f(1)` prendra la valeur :
- ☐ 0
  - ☐ 5
  - ☐ 4
  - ☐ 1
19. Un bit est :
- ☐ la longueur d'un mot mémoire
  - ☐ un battement d'horloge processeur
  - ☐ un chiffre binaire (0 ou 1)
  - ☐ l'instruction qui met fin à un programme
20. Le code suivant :
- ```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```
- affichera :
- ☐ Mineur
  - ☐ Majeur
  - ☐ rien
  - ☐ Majeur
  - ☐ Mineur

**Barème : 1 point par réponse juste (unique) ; -0,5 point par réponse fausse. Durée : 20 minutes.**

1. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche `x = 5`
- ☐ le terminal affiche `x = 2`
- ☐ le terminal affiche "Faux"
- ☐ le terminal affiche 5

2. Vous utilisez une boucle `while` quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal

3. Quel est l'opérateur de différence en C :

- ☐ `<>`
- ☐ `≠`
- ☐ `!=`
- ☐ `!`

4. Sous unix (ou linux), la commande `cd` permet de :

- ☐ ouvrir un bureau partagé (common desktop)
- ☐ détruire un fichier
- ☐ changer de répertoire courant
- ☐ jouer de la musique
- ☐ récupérer un programme arrêté avec la commande `ab`

5. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

6. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy(p, q);`
- ☐ `x = mccarthy(n);`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy();`

7. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse sémantique
- ☐ analyse harmonique
- ☐ analyse lexicale
- ☐ analyse syntaxique

8. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(int n);`
- ☐ `int carre(2);`
- ☐ `n = carre(n);`
- ☐ `int n = carre();`

9. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ rien
- ☐ Mineur
- ☐ Mineur Majeur

10. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le `main`
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique
- ☐ un ordre quelconque

11. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1; i<5; i=i+1)`
- ☐ `for(i=0; i<=5; i=i+1)`
- ☐ `for(i=1; i<=5; i=i+1)`
- ☐ `for(i=0; i<5; i=i+1)`

12. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `A && B`
- ☐ `!(A || B) == (A && !B)`
- ☐ `(!A || B)`
- ☐ `(A == TRUE) && (B == TRUE)`

13. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :
- ☐ `int n = pgcd();`
  - ☐ `n = pgcd(int p, int q);`
  - ☐ `n = pgcd(n, 3);`
  - ☐ `int pgcd(2);`
14. Pour l'extrait de programme suivant :
- ```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```
- La valeur affichée est :
- ☐ 4
  - ☐ 16
  - ☐ 8
  - ☐ 0
15. Avant de faire appel à une fonction il est nécessaire de :
- ☐ l'avoir déclarée
  - ☐ l'avoir définie
  - ☐ l'avoir déclarée et définie
  - ☐ avoir défini une constante symbolique de la taille de cette fonction

16. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?
- ☐ une accolade manquante
  - ☐ une accolade en trop
  - ☐ un point-virgule en trop
  - ☐ un point-virgule manquant
17. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :
- ☐ `kwrite TP4`
  - ☐ `mkdir TP4`
  - ☐ `new TP4`
  - ☐ `yppasswd`
18. Soit un programme contenant les lignes suivantes :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```
- qu'est ce qui sera affiché par ce printf ?
- ☐ `j = 0`
  - ☐ `j = %d`

- ☐ `j = 5`
  - ☐ `j = 4`
19. L'écriture 101 en binaire correspond au nombre naturel :
- ☐ 5
  - ☐ 101
  - ☐ 3
  - ☐ 4
20. Soit le programme principal suivant :
- ```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```
- appelant la fonction `f` ainsi définie :
- ```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```
- L'affichage dans le main est le suivant :
- ☐ `f(a,b)=13, a=8, b=5`
  - ☐ `f(3,5)=8, a=3, b=5`
  - ☐ `f(a,b)=8, a=8, b=5`
  - ☐ `f(a,b)=8, a=3, b=5`

**Barème : 1 points par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- Sur unix (ou linux), la commande `mkdir` permet de :
  - ☐ créer un fichier texte
  - ☐ ouvrir un fichier texte
  - ☐ créer un répertoire
  - ☐ changer de répertoire courant
- Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <studio.h>`
  - ☐ `#include <stdio.h>`
  - ☐ `#include <studlib.h>`
- Un enregistrement permet de grouper plusieurs valeurs dans :
  - ☐ ses chants
  - ☐ ses champs
  - ☐ ses cases
  - ☐ ses blocs
- Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 3 0 1 2
- ☐ 0 0 1 1 2 2 3

- Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
 Undefined symbols : "\_printf" ou  
 référence indéfinie vers « printf »
  - ☐ l'analyse des entrées clavier
  - ☐ l'édition de liens
  - ☐ l'analyse sémantique
  - ☐ l'analyse harmonique
- On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?
  - ☐ A && B
  - ☐ !(A || B) == (A && !B)
  - ☐ (A == TRUE) && (B == TRUE)
  - ☐ (!A || B)
- Après exécution jusqu'à la ligne 14 du programme C :
 

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

  - ☐ le terminal affiche "Faux"
  - ☐ le terminal affiche 5
  - ☐ le terminal affiche x = 2
  - ☐ le terminal affiche x = 5
- L'ordonnancement par tourniquet permet :
  - ☐ de ne pas perdre de temps avec la commutation de contexte
  - ☐ d'afficher des ronds colorés à l'écran
  - ☐ d'entretenir l'illusion que les processus tournent en parallèle
  - ☐ de doubler la mémoire disponible

- Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5

- Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a{n, x} == b{n, x}
- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a == b
- ☐ a = b

- Dans la commande gcc, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur

12. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?
- ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une directive préprocesseur **#include** manquante
  - ☐ une fonction appelée avant sa déclaration
13. Au début de la fonction **main()** on place le code :
- ```
char b = 'A';  
b = b + 2;  
printf("%c\n", b);
```
- Alors l'affichage sera :
- ☐ b
  - ☐ B
  - ☐ A
  - ☐ C
14. Quel est le problème d'un programme comportant les lignes suivantes ?
- ```
while (1)  
{  
    printf("coucou\n");  
}
```
- ☐ il ne compile pas
  - ☐ il comporte une boucle infinie
  - ☐ il n'affiche rien
  - ☐ il risque d'afficher bonjour à la place de coucou

15. Avant de faire appel à une fonction il est nécessaire de :
- ☐ l'avoir déclarée et définie
  - ☐ l'avoir définie
  - ☐ avoir défini une constante symbolique de la taille de cette fonction
  - ☐ l'avoir déclarée
16. Une variable booléenne est un variable :
- ☐ à laquelle une valeur vient d'être affectée
  - ☐ NaN (not a number, qui n'est pas un nombre)
  - ☐ réelle positive
  - ☐ jamais nulle
  - ☐ qui est vraie ou fausse
17. Le type des réels en C est :
- ☐ **real**
  - ☐ **char**
  - ☐ **double**
  - ☐ **int**
18. Pour déclarer une fonction **pgcd** qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
- ☐ **int pgcd(int y, int x);**
  - ☐ **void pgcd(int x, int y);**
  - ☐ **int pgcd(int x, y);**
  - ☐ **int pgcd(int x, int x);**

19. Soit la fonction **f** définie par :

```
int f(int a)  
{  
    printf("a = \n", %d);  
    if (a > 0)  
    {  
        return 3;  
    }  
    return 4;  
}
```

Alors l'expression **f(0)** prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

20. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ **init**
- ☐ **main**
- ☐ **begin**
- ☐ **include**

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Le type des réels en C est :
  - ☐ char
  - ☐ int
  - ☐ double
  - ☐ real
- Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max',**  
 que doit-on chercher dans le programme ?
  - ☐ une directive préprocesseur **#include** manquante
  - ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction appelée avant sa déclaration
- Dans la commande gcc, l'option **-Wall** signifie :
  - ☐ qu'on veut changer aléatoirement de fond d'écran
  - ☐ qu'il faut indenter le fichier source
  - ☐ qu'il faut lancer un débogueur
  - ☐ que l'on veut voir tous les avertissements
- Pour l'extrait de programme suivant :
 

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

 La valeur de somme affichée est :
  - ☐ 6
  - ☐ 0
  - ☐ 15
  - ☐ 10
- L'écriture 101 en binaire correspond au nombre naturel :
  - ☐ 4
  - ☐ 3
  - ☐ 5
  - ☐ 101

6. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ le programme affiche "Faux"
- ☐ la variable y vaut 5
- ☐ la variable x vaut 5 et la variable y vaut 0
- ☐ la variable x vaut 0

7. Si **carre** est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que **n** est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int n = carre();`
- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`
- ☐ `int carre(2);`

8. Si **pgcd** est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `int pgcd(2);`
- ☐ `int n = pgcd();`

9. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses champs
- ☐ ses cases
- ☐ ses blocs

10. On souhaite faire une boucle de contrôle de saisie : tant que l'entier **n** n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de **n**. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition **cond** :

- ☐ `(a<n) || (n>b)`
- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`

11. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction **main()**, alors on peut écrire en début de **main()** :

- ☐ `toto_s n, x;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `struct toto_s toto;`
- ☐ `int toto.n = 3;`

12. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal

13. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 8
- ☐ 4
- ☐ 0

14. Une variable booléenne est un variable :

- ☐ jamais nulle
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ réelle positive
- ☐ qui est vraie ou fausse
- ☐ à laquelle une valeur vient d'être affectée

15. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char c;
- ☐ char "c";
- ☐ char 'c';
- ☐ int char;

16. Au début de la fonction main() on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ cccccc
- ☐ i
- ☐ A
- ☐ ABCDEF

17. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ begin
- ☐ include
- ☐ init
- ☐ main

18. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5

19. Le langage C est un langage

- ☐ composé
- ☐ compilé
- ☐ interprété
- ☐ lu, écrit, parlé

20. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Majeur
- ☐ Mineur
- ☐ Mineur Majeur



**Barème : 1 points par réponse juste (unique); −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur **#include** manquante
- ☐ une fonction déclarée mais non définie

2. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ i
- ☐ ABCDEF
- ☐ A
- ☐ cccccc

3. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ la variable x vaut 16
- ☐ la variable x vaut  $-\frac{1}{2}$
- ☐ le programme affiche x
- ☐ le programme affiche \*\*\*\*

4. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 0 1 2 3
- ☐ 4 3 2 1

5. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un `for` dans le même programme principal

6. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur **#include** manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration

7. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 3 0 1 2
- ☐ 0 1 2 0 1 2 3
- ☐ 0 0 1 1 2 2 3

8. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `new TP4`
- ☐ `mkdir TP4`
- ☐ `yppasswd`
- ☐ `kwrite TP4`

9. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Mineur Majeur
- ☐ rien

10. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant
- ☐ une accolade en trop
- ☐ un point-virgule en trop
- ☐ une accolade manquante

11. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(2/3);`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x - 1 = racine(x);`

12. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses blocs
  - ☐ ses chants
  - ☐ ses champs
  - ☐ ses cases
13. Pour compiler un programme `prog.c`, on utilise la ligne de commande :
- ☐ `gcc -Wall prog.exe -o prog.c`
  - ☐ `gcc prog.exe -Wall -o prog.c`
  - ☐ `gcc prog.c -o -Wall prog.exe`
  - ☐ `gcc -Wall prog.c -o prog.exe`
14. Un fichier source est :
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
  - ☐ un document illisible pour les humains
  - ☐ un document qui doit être protégé
  - ☐ un document de référence du système
  - ☐ un fichier texte qui sera traduit en instructions processeur
15. Le code suivant :
- ```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 2
- ☐ 8 6 4 2 0
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2

16. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `char c;`
- ☐ `int char;`

17. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché?

- ☐ \*\*\*\*\* \*\*
- ☐ \*\* \*\*\* \*\*
- ☐ \*\*\*\*\* \*\*
- ☐ \*\* \*\* \*\*

18. Si  $x$  est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 0.5
- ☐ 0
- ☐ 1.5
- ☐ 1

19. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 0
- ☐ 5

20. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 111
- ☐ 7
- ☐ 8

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 0  
☐ 16  
☐ 4  
☐ 8

2. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction déclarée mais non définie  
☐ une fonction appelée avant sa déclaration  
☐ une directive préprocesseur **#include** manquante

3. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression **f(0)** prendra la valeur :

- ☐ 3  
☐ 0  
☐ 4

4. Pour déclarer une fonction **pgcd** qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, int x);`  
☐ `int pgcd(int y, int x);`  
☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, y);`

5. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 4  
☐ 5  
☐ 0  
☐ 1

6. Une variable booléenne est un variable :

- ☐ à laquelle une valeur vient d'être affectée  
☐ qui est vraie ou fausse  
☐ réelle positive  
☐ NaN (not a number, qui n'est pas un nombre)  
☐ jamais nulle

7. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0  
☐ 4 3 2 1  
☐ 0 1 2 3 4  
☐ 0 1 2 3

8. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction appelée avant sa déclaration  
☐ une fonction déclarée mais non définie  
☐ une directive préprocesseur **#include** manquante

9. L'écriture **101** en binaire correspond au nombre naturel :

- ☐ 5  
☐ 4  
☐ 3  
☐ 101

10. Vous utilisez une boucle **while** quand :

- ☐ vous n'avez pas déclaré de fonction  
☐ vous avez déjà fait un **for** dans le même programme principal  
☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

11. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8  
☐ 0 2 4 6  
☐ 0 1 2 3 4 5 6 7  
☐ 0 1 2 3 4 5 6

12. Quel est le problème d'un programme comportant les lignes suivantes ?
- ```
while (1)
{
    printf("coucou\n");
}
```
- ☐ il n'affiche rien
  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il comporte une boucle infinie
  - ☐ il ne compile pas
13. Les lignes
- ```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```
- ☐ ne comportent aucune erreur
  - ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
  - ☐ comportent une erreur qui ne sera pas détectée
  - ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
14. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
- ☐ un débogueur
  - ☐ `printf("Valeur de n ? %g\n", n);`
  - ☐ `printf("Valeur de n ? %d\n", n);`
  - ☐ `scanf("%d", &n);`
15. Pour compiler un programme `prog.c`, on utilise la ligne de commande :
- ☐ `gcc -Wall prog.c -o prog.exe`
  - ☐ `gcc -Wall prog.exe -o prog.c`
  - ☐ `gcc prog.exe -Wall -o prog.c`
  - ☐ `gcc prog.c -o -Wall prog.exe`

16. Le code suivant :
- ```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```
- affichera :
- ☐ Majeur
  - ☐ Mineur
  - ☐ rien
  - ☐ Mineur
17. Le bus système sert à :
- ☐ Arriver à l'heure en cours
  - ☐ transporter les processus du tourniquet au processeur
  - ☐ Écrire des données sur le disque dur
  - ☐ Transférer des données et intructions entre processeur et mémoire
18. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10 int main() {
11     int x = 5;
12     int y = 3;
13
14     x = y;
15
16     ...
17 }
```
- ☐ la variable x vaut 3

- ☐ le programme affiche "Faux"
  - ☐ la variable y vaut 5
  - ☐ la variable x vaut 5 et la variable y vaut 3
19. Soit un programme contenant les lignes suivantes :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```
- qu'est ce qui sera affiché ?
- ☐ 0 1 0 1 0 1 0 1
  - ☐ 0 0 0 1 1 1
  - ☐ 0 1 2 0 1 2
  - ☐ 1 2 1 2 3
20. Un programme en langage C doit comporter une et une seule définition de la fonction :
- ☐ `init`
  - ☐ `main`
  - ☐ `begin`
  - ☐ `include`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
  - ☐ `void pgcd(int x, int y);`
  - ☐ `int pgcd(int x, y);`
  - ☐ `int pgcd(int x, int x);`
  - ☐ `int pgcd(int y, int x);`
- Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :
  - ☐ retourner un bloc
  - ☐ répéter un bloc tant qu'une condition est vérifiée
  - ☐ sélectionner entre deux blocs à l'aide d'une condition
  - ☐ mettre les blocs en séquence les uns à la suite des autres
- Dans la commande `gcc`, l'option `-Wall` signifie :
  - ☐ qu'il faut lancer un débogueur
  - ☐ qu'il faut indenter le fichier source
  - ☐ qu'on veut changer aléatoirement de fond d'écran
  - ☐ que l'on veut voir tous les avertissements
- Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :
  - ☐ `x - 1 = racine(x);`
  - ☐ `x = racine(2/3);`
  - ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x = racine(x * x) - racine(x);`
- Au début de la fonction `main()` on place le code :
 

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

 Alors l'affichage sera :
  - ☐ b
  - ☐ A
  - ☐ B
  - ☐ C

- Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
  - ☐ alphabétique
  - ☐ dans lequel ces fonctions sont appelées dans le `main`
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ un ordre quelconque
- Après exécution jusqu'à la ligne 15 du programme C :
 

```
10 int main() {
11     int x = 5;
12     int y = 3;
13
14     x = y;
15
16     ...
17 }
```

  - ☐ la variable `x` vaut 5 et la variable `y` vaut 3
  - ☐ la variable `y` vaut 5
  - ☐ le programme affiche "Faux"
  - ☐ la variable `x` vaut 3
- Au début de la fonction `main()` on place le code :
 

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
```

 Alors l'affichage sera :
  - ☐ ABCDEF
  - ☐ cccccc
  - ☐ i
  - ☐ A
- Un enregistrement permet de grouper plusieurs valeurs dans :
  - ☐ ses cases
  - ☐ ses blocs
  - ☐ ses champs
  - ☐ ses chants

- Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 4
- ☐ 0
- ☐ 8

- On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?
  - ☐ `!(A || B) == (A && !B)`
  - ☐ `A && B`
  - ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `(!A || B)`
- Le code suivant :
 

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

 affichera :
  - ☐ 6
  - ☐ 1
  - ☐ 42
  - ☐ 0

13. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

14. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou

15. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6
- ☐ 0 2 4 6 8
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6

16. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\*
- ☐ \*\* \*\*\*
- ☐ \*\* \*\* \*
- ☐ \*\*\*\*

17. Si cette erreur apparaît à la compilation :

Undefined symbols : "\_printf" ou  
référence indéfinie vers « printf » que doit-on chercher dans le programme ?

- ☐ une faute de frappe dans un appel de fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une variable non déclarée
- ☐ un caractère interdit en C

18. Le bus système sert à :

- ☐ Arriver à l'heure en cours
- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Écrire des données sur le disque dur

19. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 111
- ☐ 7
- ☐ 8

20. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur  
Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ répéter un bloc tant qu'une condition est vérifiée
- ☐ retourner un bloc
- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ mettre les blocs en séquence les uns à la suite des autres

2. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0; i<=5; i=i+1)`
- ☐ `for(i=0; i<5; i=i+1)`
- ☐ `for(i=1; i<5; i=i+1)`
- ☐ `for(i=1; i<=5; i=i+1)`

3. Après exécution jusqu'à la ligne 15 du programme C :

```

10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

- ☐ la variable x vaut 5 et la variable y vaut 3
- ☐ la variable x vaut 3
- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"

4. Soit la fonction f définie par :

```

int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
}
```

```

}
return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 5
- ☐ 1

5. Vous utilisez une boucle `while` quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

6. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.exe -o prog.c`

7. Si a et b sont deux variables de type :

```

struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ `a == b`
- ☐ `a = b`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a{n, x} == b{n, x}`

8. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse sémantique
- ☐ analyse lexicale
- ☐ analyse harmonique
- ☐ analyse syntaxique

9. Soit le programme principal suivant :

```

int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n", f(a,b), a, b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```

int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

10. Soit un programme contenant les lignes suivantes :

```

int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 1 2 1 2 3
- ☐ 0 0 0 1 1 1
- ☐ 0 1 2 0 1 2
- ☐ 0 1 0 1 0 1 0 1

11. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `struct toto_s toto;`
- ☐ `toto_s n, x;`
- ☐ `int toto.n = 3;`

12. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b
- ☐ B
- ☐ A
- ☐ C

13. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`

14. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`
- ☐ `int factorielle(double n);`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle();`

15. On considère deux variables booléennes A et B initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `A && B`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `(!A || B)`
- ☐ `!(A || B) == (A && !B)`

16. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 3
- ☐ 4

17. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`

18. Un registre du processeur est :

- ☐ un composant qui contient la liste des fichiers du système
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

19. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses champs
- ☐ ses blocs
- ☐ ses cases

20. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a == b
- ☐ a{n, x} == b{n, x}
- ☐ (a.n == b.n) && (a.x == b.x)
- ☐ a = b

2. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ int factorielle(int x);
- ☐ struct int factorielle(int n);
- ☐ int factorielle();
- ☐ int factorielle(double n);

3. Sous unix (ou linux), la commande **ls** permet de :

- ☐ afficher le contenu d'un fichier texte
- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ voir des clips musicaux
- ☐ compiler un programme

4. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

Undefined symbols : "\_printf" ou  
référence indéfinie vers « printf »

- ☐ l'édition de liens
- ☐ l'analyse harmonique
- ☐ l'analyse sémantique
- ☐ l'analyse des entrées clavier

5. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien

6. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 2
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2 0
- ☐ 8 6 4 2

7. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

8. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 3

9. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti

10. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ int char;
- ☐ char c;
- ☐ char 'c';
- ☐ char "c";

11. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses blocs
- ☐ ses chants
- ☐ ses cases

12. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

```
☐ int struct toto_s = {3, -1e10};
☐ struct toto_s toto;
☐ toto_s struct z = {3, 0.5};
☐ toto_s n, x;
☐ int toto.n = 3;
```

13. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

```
☐ 1 2 3 1 2
☐ 0 0 1 1 2 2
☐ 0 1 2 0 1 2
☐ 0 1 0 1 0 1
```

14. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

```
☐ un débogueur
☐ printf("Valeur de n ? %d\n", n);
☐ scanf("%d", &n);
☐ printf("Valeur de n ? %g\n", n);
```

15. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

```
☐ int pgcd(int x, y);
☐ int pgcd(int x, int x);
☐ void pgcd(int x, int y);
☐ int pgcd(int y, int x);
```

16. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

```
☐ void exposant(double x^n);
☐ exposant(double x, int n, int r);
☐ double exposant(double x, int n);
☐ int exposant(double n, int x);
```

17. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

```
☐ 7
☐ 0
☐ 5
```

18. Le code suivant :

```
int age = 20;
if (age < 18)
{
```

```
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

```
☐ Mineur
☐ Majeur
☐ Mineur
    Majeur
☐ rien
```

19. Un bit est :

```
☐ un chiffre binaire (0 ou 1)
☐ l'instruction qui met fin à un programme
☐ la longueur d'un mot mémoire
☐ un battement d'horloge processeur
```

20. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

```
☐ 4 3 2 1 0
☐ 0 1 2 3 4
☐ 1 2 3 4
☐ 4 3 2 1
```

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée

2. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 1 2 1 2 3
- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
- ☐ 0 0 0 1 1 1

3. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 4
- ☐ 0
- ☐ 8

4. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

5. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5

6. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu(int char);`
- ☐ `double afficher_menu();`
- ☐ `int afficher_menu();`
- ☐ `char afficher_menu printf("menu");`
- ☐ `void afficher_menu();`

7. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char c;`

8. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=x et y=y\n");`

9. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche \*\*\*\*
- ☐ la variable x vaut 16
- ☐ la variable x vaut  $-\frac{1}{2}$
- ☐ le programme affiche x

10. Une variable booléenne est un variable :

- ☐ jamais nulle
- ☐ réelle positive
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ à laquelle une valeur vient d'être affectée
- ☐ qui est vraie ou fausse

11. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1

12. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int x, int x);`

13. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou

14. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse sémantique
- ☐ analyse lexicale
- ☐ analyse syntaxique

15. Le type des réels en C est :

- ☐ `double`
- ☐ `int`
- ☐ `real`
- ☐ `char`

16. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 1
- ☐ 4

17. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`
- ☐ `double exposant(double x, int n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`

18. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `!(A || B) == (A && !B)`
- ☐ `A && B`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `(!A || B)`

19. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 0 1 2 3
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0

20. Un fichier source est :

- ☐ un document qui doit être protégé
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un document de référence du système
- ☐ un document illisible pour les humains
- ☐ un fichier texte qui sera traduit en instructions processeur

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Quel est l'opérateur de différence en C :

- ☐ !  
☐ <>  
☐ !=  
☐ ≠

2. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0; i<=5; i=i+1)`  
☐ `for(i=1; i<5; i=i+1)`  
☐ `for(i=1; i<=5; i=i+1)`  
☐ `for(i=0; i<5; i=i+1)`

3. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8  
☐ 0 1 2 3 4 5 6  
☐ 0 2 4 6  
☐ 0 1 2 3 4 5 6 7

4. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main  
☐ alphabétique  
☐ dans lequel vous avez déclaré ces fonction  
☐ un ordre quelconque

5. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 1 2 1 2 3  
☐ 0 1 2 0 1 2  
☐ 0 1 0 1 0 1 0 1  
☐ 0 0 0 1 1 1

6. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle(p, q);`  
☐ `printf("%d", factorielle(n));`  
☐ `int factorielle(int 2);`  
☐ `n = factorielle();`

7. Le bus système sert à :

- ☐ Écrire des données sur le disque dur  
☐ Transférer des données et intructions entre processeur et mémoire  
☐ transporter les processus du tourniquet au processeur  
☐ Arriver à l'heure en cours

8. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`  
☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, int x);`  
☐ `int pgcd(int x, y);`

9. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur  
Majeur  
☐ Mineur  
☐ Majeur  
☐ rien

10. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3  
☐ 0  
☐ 4

11. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée et définie  
☐ l'avoir déclarée  
☐ avoir défini une constante symbolique de la taille de cette fonction  
☐ l'avoir définie

12. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 2
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2 0
- ☐ 8 6 4 2

13. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0

14. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(int p, int q);`
- ☐ `n = pgcd(n, 3);`
- ☐ `int n = pgcd();`
- ☐ `int pgcd(2);`

15. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ `j = %d`
- ☐ `j = 4`
- ☐ `j = 0`
- ☐ `j = 5`

16. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s n, x;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `int toto.n = 3;`
- ☐ `struct toto_s toto;`

17. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses blocs
- ☐ ses champs
- ☐ ses chants

18. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `char c;`
- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char 'c';`

19. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `int exposant(double n, int x);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `void exposant(double x^n);`
- ☐ `double exposant(double x, int n);`

20. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché?

- ☐ 0 1 0 1 0 1
- ☐ 1 2 3 1 2
- ☐ 0 1 2 0 1 2
- ☐ 0 0 1 1 2 2

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un fichier source est :

- ☐ un document illisible pour les humains
- ☐ un document qui doit être protégé
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document de référence du système
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur

2. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `int saisie_utilisateur();`

3. Avant de faire appel à une fonction il est nécessaire de :

- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir définie
- ☐ l'avoir déclarée
- ☐ l'avoir déclarée et définie

4. Le langage C est un langage

- ☐ compilé
- ☐ interprété
- ☐ composé
- ☐ lu, écrit, parlé

5. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
```

```
}
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 4
- ☐ j = 5
- ☐ j = 0
- ☐ j = %d

6. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy(p, q);`
- ☐ `n = mccarthy();`
- ☐ `int mccarthy(int 2);`
- ☐ `x = mccarthy(n);`

7. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ i
- ☐ ABCDEF
- ☐ A
- ☐ cccccc

8. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char "c";`

9. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

10. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7
- ☐ 111
- ☐ 8
- ☐ 3

11. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\* \*\*\* \*\*\*\* \*\*\*\*\*
- ☐ \*\*\*\* \*\*\*\*\* \*\*
- ☐ \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*
- ☐ \*\*\*\*\* \*\* \*\*

12. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
- ☐ `n = factorielle();`
  - ☐ `n = factorielle(p, q);`
  - ☐ `printf("%d", factorielle(n));`
  - ☐ `int factorielle(int 2);`
13. Dans la commande gcc, l'option `-Wall` signifie :
- ☐ qu'on veut changer aléatoirement de fond d'écran
  - ☐ qu'il faut lancer un débogueur
  - ☐ que l'on veut voir tous les avertissements
  - ☐ qu'il faut indenter le fichier source
14. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.
- ☐ `#define taille = N`
  - ☐ `#define taille = 3`
  - ☐ `#define N 3`
  - ☐ `#define N = 3`
15. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
- ☐ `#appart <stdlib.h>`
  - ☐ `#include <studlib.h>`
  - ☐ `#include <stdio.h>`
  - ☐ `#include <studio.h>`
16. Si le code :
- ```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s n, x;`
  - ☐ `int struct toto_s = {3, -1e10};`
  - ☐ `struct toto_s toto;`
  - ☐ `int toto.n = 3;`
  - ☐ `toto_s struct z = {3, 0.5};`
17. Une *segmentation fault* est une erreur qui survient lorsque :
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
  - ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
  - ☐ la division du programme en zones homogènes échoue
  - ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
18. Le code suivant :
- ```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```
- affichera :
- ☐ 1

- ☐ 0
- ☐ 6
- ☐ 42

19. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ `int k;`
- ☐ `loop i;`
- ☐ `int %d;`
- ☐ `int loop n;`

20. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 0
- ☐ 5



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42  
☐ 0  
☐ 6  
☐ 1

2. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y = 3;
13
14     x = y;
15
16     ...
17 }
```

- ☐ la variable x vaut 5 et la variable y vaut 3  
☐ la variable x vaut 3  
☐ la variable y vaut 5  
☐ le programme affiche "Faux"

3. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 4  
☐ 3  
☐ 0

4. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5  
☐ f(a,b)=8, a=3, b=5  
☐ f(3,5)=8, a=3, b=5  
☐ f(a,b)=13, a=8, b=5

5. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main  
☐ dans lequel vous avez déclaré ces fonction  
☐ alphabétique  
☐ un ordre quelconque

6. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2  
☐ 0 2 4 6 8  
☐ 8 6 4 2 0  
☐ 8 2

7. Si cette erreur apparaît à la compilation :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ une variable non déclarée  
☐ une faute de frappe dans un appel de fonction  
☐ une directive préprocesseur #include manquante  
☐ un caractère interdit en C

8. Vous utilisez une boucle while quand :

- ☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous n'avez pas déclaré de fonction  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ vous avez déjà fait un for dans le même programme principal

9. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ int exposant(double n, int x);  
☐ exposant(double x, int n, int r);  
☐ double exposant(double x, int n);  
☐ void exposant(double x^n);

10. Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :

- ☐ void afficher\_menu();  
☐ double afficher\_menu();  
☐ char afficher\_menu(printf("menu"));  
☐ int afficher\_menu();  
☐ int afficher\_menu(int char);

11. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :
- ☐ mkdir TP4
  - ☐ new TP4
  - ☐ kwrite TP4
  - ☐ yppasswd
12. Le code suivant :
- ```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 4 3 2 1
  - ☐ 0 1 2 3
  - ☐ 0 1 2 3 4
  - ☐ 4 3 2 1 0
13. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »**
- ☐ l'analyse sémantique
  - ☐ l'analyse harmonique
  - ☐ l'édition de liens
  - ☐ l'analyse des entrées clavier
14. Le type des réels en C est :
- ☐ int
  - ☐ real
  - ☐ double
  - ☐ char

15. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```
- ☐ le programme affiche \*\*\*\*
  - ☐ la variable x vaut 16
  - ☐ la variable x vaut  $-\frac{1}{2}$
  - ☐ le programme affiche x
16. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max' , que doit-on chercher dans le programme ?**
- ☐ une fonction appelée avant sa déclaration
  - ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une directive préprocesseur **#include** manquante
17. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?
- ☐ (!A || B)
  - ☐ !(A || B) == (A && !B)
  - ☐ (A == TRUE) && (B == TRUE)
  - ☐ A && B

18. Le code suivant :
- ```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 0 1 2 3 4 5 6 7
  - ☐ 0 2 4 6 8
  - ☐ 0 2 4 6
  - ☐ 0 1 2 3 4 5 6
19. Sur unix (ou linux), la commande **mkdir** permet de :
- ☐ créer un répertoire
  - ☐ créer un fichier texte
  - ☐ changer de répertoire courant
  - ☐ ouvrir un fichier texte
20. Soit la fonction g définie par :
- ```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```
- Alors l'expression **g(0)** prendra la valeur :
- ☐ 0
  - ☐ 7
  - ☐ 5

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2  
☐ 1 2 3 1 2  
☐ 0 1 2 0 1 2  
☐ 0 1 0 1 0 1

2. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »**

- ☐ l'analyse harmonique  
☐ l'analyse sémantique  
☐ l'analyse des entrées clavier  
☐ l'édition de liens

3. Dans la commande gcc, l'option **-Wall** signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran  
☐ qu'il faut indenter le fichier source  
☐ qu'il faut lancer un débogueur  
☐ que l'on veut voir tous les avertissements

4. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8  
☐ 8 6 4 2 0  
☐ 8 2  
☐ 8 6 4 2

5. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`  
☐ `for(i=0;i<=5;i=i+1)`  
☐ `for(i=1;i<=5;i=i+1)`  
☐ `for(i=0;i<5;i=i+1)`

6. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0  
☐ 0 1 2 3 4  
☐ 1 2 3 4  
☐ 4 3 2 1

7. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`  
☐ `int exposant(double n, int x);`  
☐ `double exposant(double x, int n);`  
☐ `void exposant(double x^n);`

8. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A  
☐ i  
☐ ABCDEF  
☐ cccccc

9. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5  
☐ 0  
☐ 7

10. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche `x = 5`  
☐ le terminal affiche "Faux"  
☐ le terminal affiche `x = 2`  
☐ le terminal affiche 5

11. Pour déclarer une fonction **pgcd** qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`  
☐ `int pgcd(int y, int x);`  
☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, int x);`

12. Le type des réels en C est :
- ☐ `char`
  - ☐ `double`
  - ☐ `real`
  - ☐ `int`
13. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ dans lequel vous avez déclaré ces fonction
  - ☐ dans lequel ces fonctions sont appelées dans le `main`
  - ☐ un ordre quelconque
  - ☐ alphabétique
14. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :
- ☐ `void afficher_date(date_s d);`
  - ☐ `struct date_s afficher_date(struct date_s d);`
  - ☐ `void afficher_date(struct date_s d);`
  - ☐ `int afficher_date(date_s d);`
15. Vous utilisez une boucle `while` quand :
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un `for` dans le même programme principal
16. Soit un programme contenant les lignes suivantes :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```
- qu'est ce qui sera affiché ?
- ☐ 0 1 2 0 1 2
  - ☐ 1 2 1 2 3
  - ☐ 0 0 0 1 1 1
  - ☐ 0 1 0 1 0 1 0 1
17. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :
- ☐ `x = racine(x * x) - racine(x);`
  - ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x = racine(2/3);`
  - ☐ `x - 1 = racine(x);`
18. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :
- ☐ `kwrite TP4`
  - ☐ `new TP4`
  - ☐ `mkdir TP4`
  - ☐ `yppasswd`
19. Au début de la fonction `main()` on place le code :
- ```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```
- Alors l'affichage sera :
- ☐ b
  - ☐ C
  - ☐ B
  - ☐ A
20. Si cette erreur apparaît à la compilation : `error: expected ';' before '}' token` que doit-on chercher dans le programme ?
- ☐ une accolade manquante
  - ☐ une accolade en trop
  - ☐ un point-virgule manquant
  - ☐ un point-virgule en trop

**Barème : 1 point par réponse juste (unique) ; -0,5 point par réponse fausse. Durée : 20 minutes.**

1. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0; i<=5; i=i+1)`  
☐ `for(i=1; i<=5; i=i+1)`  
☐ `for(i=1; i<5; i=i+1)`  
☐ `for(i=0; i<5; i=i+1)`

2. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, int x);`  
☐ `int pgcd(int x, y);`  
☐ `int pgcd(int y, int x);`

3. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0  
☐ 4  
☐ 3

4. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
```

```
}
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ `j = %d`  
☐ `j = 5`  
☐ `j = 0`  
☐ `j = 4`

5. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n", f(a,b), a, b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`  
☐ `f(a,b)=8, a=3, b=5`  
☐ `f(a,b)=8, a=8, b=5`  
☐ `f(3,5)=8, a=3, b=5`

6. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ vous n'avez pas déclaré de fonction  
☐ vous avez déjà fait un `for` dans le même programme principal  
☐ l'incrément de la variable de boucle n'est pas 1

7. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.c -o prog.exe`  
☐ `gcc prog.exe -Wall -o prog.c`  
☐ `gcc -Wall prog.exe -o prog.c`  
☐ `gcc prog.c -o -Wall prog.exe`

8. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une directive préprocesseur `#include` manquante  
☐ une fonction déclarée mais non définie  
☐ une fonction appelée avant sa déclaration

9. Si cette erreur apparaît à la compilation :  
**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ un caractère interdit en C  
☐ une variable non déclarée  
☐ une directive préprocesseur `#include` manquante  
☐ une faute de frappe dans un appel de fonction

10. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`  
☐ `#appart <stdlib.h>`  
☐ `#include <stdio.h>`  
☐ `#include <studio.h>`

11. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche `x = 2`  
☐ le terminal affiche "Faux"  
☐ le terminal affiche 5  
☐ le terminal affiche `x = 5`

12. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur

13. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `scanf("%d", &n);`
- ☐ un débogueur

14. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ mettre les blocs en séquence les uns à la suite des autres

☐ retourner un bloc

☐ répéter un bloc tant qu'une condition est vérifiée

15. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `printf("%d", factorielle(n));`
- ☐ `n = factorielle(p, q);`
- ☐ `int factorielle(int 2);`
- ☐ `n = factorielle();`

16. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1

17. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
- ☐ `(a<n) || (n>b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<=n) && (n<=b)`

18. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `int char;`

19. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

20. Sous unix (ou linux), la commande `cd` permet de :

- ☐ détruire un fichier
- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ jouer de la musique
- ☐ changer de répertoire courant
- ☐ ouvrir un bureau partagé (common desktop)

**Barème : 1 points par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`  
☐ `int n = pgcd();`  
☐ `int pgcd(2);`  
☐ `n = pgcd(int p, int q);`

2. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=13, a=8, b=5`  
☐ `f(a,b)=8, a=3, b=5`  
☐ `f(3,5)=8, a=3, b=5`

3. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction  
☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous avez déjà fait un `for` dans le même programme principal  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

4. Un fichier source est :

- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur  
☐ un document qui doit être protégé  
☐ un document illisible pour les humains  
☐ un document de référence du système  
☐ un fichier texte qui sera traduit en instructions processeur

5. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable `x` vaut 5 et la variable `y` vaut 0  
☐ la variable `x` vaut 0  
☐ le programme affiche "Faux"  
☐ la variable `y` vaut 5

6. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.c -o prog.exe`  
☐ `gcc prog.exe -Wall -o prog.c`  
☐ `gcc prog.c -o -Wall prog.exe`  
☐ `gcc -Wall prog.exe -o prog.c`

7. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction  
☐ alphabétique  
☐ dans lequel ces fonctions sont appelées dans le `main`  
☐ un ordre quelconque

8. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine  
☐ certains programmes sont de vrais plats de spaghetti  
☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée  
☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine

9. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`  
☐ `void saisie_utilisateur(char c);`  
☐ `int saisie_utilisateur();`  
☐ `void saisie_utilisateur(int n);`

10. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop  
☐ un point-virgule manquant  
☐ une accolade en trop  
☐ une accolade manquante

11. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle();`  
☐ `int factorielle(double n);`  
☐ `int factorielle(int x);`  
☐ `struct int factorielle(int n);`

12. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
```

```

5  add r2 r0
6  ecriture r0 8
7  stop
8  5

```

- ☐ la case mémoire 8 contiendra 16
- ☐ la case mémoire 8 contiendra 0
- ☐ le terminal affiche 8
- ☐ le bus explose

13. Pour l'extrait de programme suivant :

```

int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);

```

La valeur affichée est :

- ☐ 8
- ☐ 16
- ☐ 4
- ☐ 0

14. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ new TP4
- ☐ kwrite TP4
- ☐ mkdir TP4
- ☐ yppasswd

15. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `scanf("%d", &n);`
- ☐ un débogueur
- ☐ `printf("Valeur de n ? %g\n", n);`

16. Pour l'extrait de programme suivant :

```

int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}

```

qu'est ce qui sera affiché ?

- ☐ \*\* \*\*\* \*\*\*\*\*
- ☐ \*\*\*\* \*\*\*\*\*
- ☐ \*\*\*\*\* \*\*\* \*\*
- ☐ \*\* \*\* \*\* \*\*

17. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

Undefined symbols : "\_printf" ou  
référence indéfinie vers « printf »

- ☐ l'analyse harmonique
- ☐ l'analyse des entrées clavier
- ☐ l'analyse sémantique
- ☐ l'édition de liens

18. Si le code :

```

struct toto_s
{
    int n;
    double x;
};

```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `struct toto_s toto;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `toto_s n, x;`
- ☐ `int toto.n = 3;`

19. Quel est le problème d'un programme comportant les lignes suivantes ?

```

while (1)
{
    printf("coucou\n");
}

```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie

20. Soit la fonction `g` définie par :

```

int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}

```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`  
☐ `n = pgcd(n, 3);`  
☐ `n = pgcd(int p, int q);`  
☐ `int pgcd(2);`

2. Avant de faire appel à une fonction il est nécessaire de :

- ☐ avoir défini une constante symbolique de la taille de cette fonction  
☐ l'avoir déclarée  
☐ l'avoir définie  
☐ l'avoir déclarée et définie

3. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`  
☐ `f(3,5)=8, a=3, b=5`  
☐ `f(a,b)=8, a=3, b=5`  
☐ `f(a,b)=8, a=8, b=5`

4. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable `x` vaut 0  
☐ la variable `x` vaut 5 et la variable `y` vaut 0  
☐ la variable `y` vaut 5  
☐ le programme affiche "Faux"

5. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur  
☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal  
☐ la division du programme en zones homogènes échoue  
☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée

6. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(A == TRUE) && (B == TRUE)`  
☐ `(!A || B)`  
☐ `A && B`  
☐ `!(A || B) == (A && !B)`

7. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3  
☐ 111  
☐ 8  
☐ 7

8. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants  
☐ ses cases  
☐ ses blocs  
☐ ses champs

9. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4  
☐ 4 3 2 1  
☐ 4 3 2 1 0  
☐ 0 1 2 3

10. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ chacun son tour, après que le processus précédent a terminé  
☐ tous ensemble  
☐ en parallèle, chacun dans un registre  
☐ tour à tour, un petit peu à chaque fois

11. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 5
- ☐ j = %d
- ☐ j = 0
- ☐ j = 4

12. Si x est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1
- ☐ 0.5
- ☐ 1.5
- ☐ 0

13. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

14. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(int n);`

15. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n,x,y");`

16. Vous utilisez une boucle while quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un for dans le même programme principal

17. Au début de la fonction main() on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ cccccc
- ☐ A
- ☐ ABCDEF
- ☐ i

18. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ `int k;`
- ☐ `int loop n;`
- ☐ `int %d;`
- ☐ `loop i;`

19. Un registre du processeur est :

- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une gamme de fréquence de fonctionnement du processeur

20. Le langage C est un langage

- ☐ lu, écrit, parlé
- ☐ compilé
- ☐ interprété
- ☐ composé

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui ne sera pas détectée
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique

2. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int n = carre();`
- ☐ `int carre(2);`
- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`

3. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n,x,y");`

4. Si cet avertissement apparaît à la compilation : `warning: implicit declaration of function 'max', que doit-on chercher dans le programme ?`

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration
- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante

5. Si cette erreur apparaît à la compilation : `error: expected ';' before '}' token` que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant
- ☐ un point-virgule en trop
- ☐ une accolade manquante
- ☐ une accolade en trop

6. Quels calculs peut-on programmer en programmation structurée ?

- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti

7. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ rien

8. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur
- ☐ qu'on veut changer aléatoirement de fond d'écran

9. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction

10. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Majeur
- ☐ Mineur Majeur
- ☐ Mineur

11. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle();`
- ☐ `printf("%d", factorielle(n));`
- ☐ `int factorielle(int 2);`
- ☐ `n = factorielle(p, q);`

12. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 0 1 2 3

13. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `struct toto_s toto;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `toto_s n, x;`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `int toto.n = 3;`

14. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a{n, x} == b{n, x}`
- ☐ `a == b`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a = b`

15. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`
- ☐ `int exposant(double n, int x);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `double exposant(double x, int n);`

16. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction

17. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
- ☐ 1 2 1 2 3
- ☐ 0 0 0 1 1 1

18. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(char c);`

19. Le type des réels en C est :

- ☐ `double`
- ☐ `real`
- ☐ `char`
- ☐ `int`

20. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `int mccarthy(int 2);`
- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy();`
- ☐ `n = mccarthy(p, q);`

2. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

3. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

- ☐ 0
- ☐ 0.5
- ☐ 1
- ☐ 1.5

4. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%x et y=%y\n");`

5. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A
- ☐ i
- ☐ cccccc
- ☐ ABCDEF

6. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction

7. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int pgcd(2);`
- ☐ `int n = pgcd();`
- ☐ `n = pgcd(n, 3);`
- ☐ `n = pgcd(int p, int q);`

8. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal

9. Vous utilisez une boucle `while` quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un `for` dans le même programme principal

10. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou

11. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 3
- ☐ 20
- ☐ 16

12. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade en trop
- ☐ un point-virgule en trop
- ☐ un point-virgule manquant
- ☐ une accolade manquante

13. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 5
- ☐ 1

14. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2
- ☐ 0 1 0 1 0 1
- ☐ 1 2 3 1 2
- ☐ 0 1 2 0 1 2

15. Un fichier source est :

- ☐ un document illisible pour les humains
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document qui doit être protégé
- ☐ un document de référence du système

16. Un registre du processeur est :

- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

17. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ mkdir TP4
- ☐ yppasswd
- ☐ new TP4
- ☐ kwrite TP4

18. Au début de la fonction main() on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C
- ☐ B
- ☐ b
- ☐ A

19. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0

20. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3
- ☐ 4 3 2 1

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4
- ☐ 1
- ☐ 0
- ☐ 5

2. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un fichier texte
- ☐ changer de répertoire courant
- ☐ créer un répertoire
- ☐ ouvrir un fichier texte

3. Sous unix (ou linux), la commande `cd` permet de :

- ☐ ouvrir un bureau partagé (common desktop)
- ☐ jouer de la musique
- ☐ détruire un fichier
- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ changer de répertoire courant

4. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

5. Un registre du processeur est :

- ☐ un composant qui contient la liste des fichiers du système
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une unité de calcul spécialisée de l'ordinateur

6. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4

7. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C
- ☐ B
- ☐ b
- ☐ A

8. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante

9. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 0
- ☐ 6
- ☐ 1
- ☐ 42

10. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`

11. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `int char;`

12. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

☐ `exposant(double x, int n, int r);`  
☐ `void exposant(double x^n);`  
☐ `double exposant(double x, int n);`  
☐ `int exposant(double n, int x);`

13. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=13, a=8, b=5`  
☐ `f(3,5)=8, a=3, b=5`  
☐ `f(a,b)=8, a=3, b=5`

14. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

☐ `(a<n) || (n>b)`

☐ `a<=n<=b`  
☐ `(n<=a) && (n<=b)`  
☐ `(a<=n) && (n<=b)`

15. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

☐ `x = racine(racine(x)*racine(x));`  
☐ `x = racine(2/3);`  
☐ `x = racine(x * x) - racine(x);`  
☐ `x - 1 = racine(x);`

16. Dans la commande gcc, l'option `-Wall` signifie :

☐ qu'on veut changer aléatoirement de fond d'écran  
☐ qu'il faut indenter le fichier source  
☐ qu'il faut lancer un débogueur  
☐ que l'on veut voir tous les avertissements

17. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

☐ `**** * * * * *`  
☐ `** * * * * *`  
☐ `** * * * * *`  
☐ `***** * * * *`

18. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
```

```
{
    return 3;
}
return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

☐ 3  
☐ 4  
☐ 0

19. Après exécution jusqu'à la ligne 15 du programme C :

```
10 ...
11 int main() {
12     int x = 5;
13
14     x = 3 * x + 1;
15
16     ...
17 }
```

☐ la variable `x` vaut 16  
☐ la variable `x` vaut  $-\frac{1}{2}$   
☐ le programme affiche `x`  
☐ le programme affiche `****`

20. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

☐ 7  
☐ 0  
☐ 5



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

☐ `saisie_utilisateur scanf(%d);`  
☐ `void saisie_utilisateur(char c);`  
☐ `int saisie_utilisateur();`  
☐ `void saisie_utilisateur(int n);`

2. Le code suivant :

```
int i;  
for (i = 0; i < 5; i = i + 1)  
{  
    printf("%d ", i);  
}  
printf("\n");
```

affichera :

☐ 0 1 2 3  
☐ 4 3 2 1 0  
☐ 4 3 2 1  
☐ 0 1 2 3 4

3. Le langage C est un langage

☐ lu, écrit, parlé  
☐ compilé  
☐ composé  
☐ interprété

4. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

`Undefined symbols : "_printf" ou référence indéfinie vers « printf »`

☐ l'analyse harmonique  
☐ l'analyse sémantique  
☐ l'analyse des entrées clavier  
☐ l'édition de liens

5. Un fichier source est :

☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur  
☐ un document illisible pour les humains  
☐ un document de référence du système  
☐ un fichier texte qui sera traduit en instructions processeur  
☐ un document qui doit être protégé

6. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...  
11    int main() {  
12        int x = 5;  
13  
14        x = 3 * x + 1;  
15  
16        ...  
17    }
```

☐ le programme affiche x  
☐ la variable x vaut 16  
☐ la variable x vaut  $-\frac{1}{2}$   
☐ le programme affiche \*\*\*\*

7. Sur unix (ou linux), la commande `mkdir` permet de :

☐ changer de répertoire courant  
☐ créer un répertoire  
☐ ouvrir un fichier texte  
☐ créer un fichier texte

8. Un enregistrement permet de grouper plusieurs valeurs dans :

☐ ses cases  
☐ ses chants  
☐ ses champs  
☐ ses blocs

9. Le type des réels en C est :

☐ `double`  
☐ `int`  
☐ `char`  
☐ `real`

10. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

☐ `printf("x=%d et y=%d\n",x,y);`  
☐ `printf("x=%d et y=%d\n",x y);`  
☐ `printf("x=x et y=y\n");`  
☐ `printf("x=%d et y=%d\n",x,y);`

11. Vous utilisez une boucle `while` quand :

☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous n'avez pas déclaré de fonction  
☐ vous avez déjà fait un `for` dans le même programme principal

12. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

☐ `for(i=1;i<=5;i=i+1)`  
☐ `for(i=0;i<5;i=i+1)`  
☐ `for(i=0;i<=5;i=i+1)`  
☐ `for(i=1;i<5;i=i+1)`

13. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)  
{  
    printf("coucou\n");  
}
```

☐ il risque d'afficher bonjour à la place de coucou  
☐ il comporte une boucle infinie  
☐ il ne compile pas  
☐ il n'affiche rien

14. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

☐ `int k;`  
☐ `int %d;`  
☐ `loop i;`  
☐ `int loop n;`

15. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

```
☐ void afficher_date(date_s d);  
☐ void afficher_date(struct date_s d);  
☐ int afficher_date(date_s d);  
☐ struct date_s afficher_date(struct date_s d);
```

16. Si le code :

```
struct toto_s  
{  
    int n;  
    double x;  
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

```
☐ toto_s n, x;  
☐ toto_s struct z = {3, 0.5};  
☐ int toto.n = 3;
```

```
☐ int struct toto_s = {3, -1e10};  
☐ struct toto_s toto;
```

17. Le code suivant :

```
int i;  
for (i = 8; i > 0; i = i - 2)  
{  
    printf("%d ", i);  
}  
printf("\n");
```

affichera :

```
☐ 8 6 4 2 0  
☐ 8 2  
☐ 0 2 4 6 8  
☐ 8 6 4 2
```

18. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

```
☐ x - 1 = racine(x);
```

```
☐ x = racine(2/3);  
☐ x = racine(x * x) - racine(x);  
☐ x = racine(racine(x)*racine(x));
```

19. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

```
☐ printf("Valeur de n ? %d\n", n);  
☐ printf("Valeur de n ? %g\n", n);  
☐ scanf("%d", &n);  
☐ un débogueur
```

20. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

```
☐ une directive préprocesseur #include manquante  
☐ une fonction appelée avant sa déclaration  
☐ une fonction déclarée mais non définie  
☐ un désaccord entre la déclaration et la définition d'une fonction
```

**Barème : 1 point par réponse juste (unique) ; –0,5 point par réponse fausse. Durée : 20 minutes.**

1. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#appart <stdlib.h>`  
☐ `#include <studlib.h>`  
☐ `#include <stdio.h>`  
☐ `#include <studio.h>`

2. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ `i`  
☐ `cccccc`  
☐ `A`  
☐ `ABCDEF`

3. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define taille = N`  
☐ `#define taille = 3`  
☐ `#define N 3`  
☐ `#define N = 3`

4. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un fichier texte  
☐ changer de répertoire courant  
☐ ouvrir un fichier texte  
☐ créer un répertoire

5. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8  
☐ 16  
☐ 0  
☐ 4

6. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration  
☐ une fonction déclarée mais non définie  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une directive préprocesseur `#include` manquante

7. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas  
☐ il n'affiche rien  
☐ il risque d'afficher bonjour à la place de coucou  
☐ il comporte une boucle infinie

8. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique  
☐ ne comportent aucune erreur  
☐ comportent une erreur qui sera détectée au cours de l'édition de lien  
☐ comportent une erreur qui ne sera pas détectée

9. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1  
☐ 0  
☐ 5  
☐ 4

10. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran  
☐ qu'il faut indenter le fichier source  
☐ que l'on veut voir tous les avertissements  
☐ qu'il faut lancer un débogueur

11. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche `x = 5`  
☐ le terminal affiche 5  
☐ le terminal affiche "Faux"  
☐ le terminal affiche `x = 2`

12. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(n<=a) && (n<=b)`
  - ☐ `(a<=n) && (n<=b)`
  - ☐ `a<=n<=b`
  - ☐ `(a<n) || (n>b)`
13. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :
- ☐ analyse lexicale
  - ☐ analyse harmonique
  - ☐ analyse sémantique
  - ☐ analyse syntaxique
14. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
- ☐ `void exposant(double x^n);`

- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`
- ☐ `exposant(double x, int n, int r);`

15. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.exe -o prog.c`

16. Si  $x$  est une variable réelle (de type `double`) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1.5
- ☐ 0
- ☐ 1
- ☐ 0.5

17. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char 'c';`

18. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int pgcd(2);`
- ☐ `int n = pgcd();`
- ☐ `n = pgcd(int p, int q);`
- ☐ `n = pgcd(n, 3);`

19. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que  $n$  est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`
- ☐ `int carre(2);`
- ☐ `int n = carre();`

20. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ B  
☐ b  
☐ C  
☐ A

2. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `printf("%d", factorielle(n));`  
☐ `int factorielle(int 2);`  
☐ `n = factorielle();`  
☐ `n = factorielle(p, q);`

3. Sous unix (ou linux), la commande `ls` permet de :

- ☐ compiler un programme  
☐ afficher la liste de fichiers contenus dans un répertoire  
☐ voir des clips musicaux  
☐ afficher le contenu d'un fichier texte

4. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3  
☐ 4  
☐ 0

5. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 20  
☐ 6  
☐ 3  
☐ 16

6. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `char tableau[5];`  
☐ `int toto[taille=5];`  
☐ `int[] new tableau(5);`  
☐ `int toto[5];`  
☐ `int tab[] = 5;`

7. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6  
☐ 0  
☐ 15  
☐ 10

8. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8  
☐ 7  
☐ 3  
☐ 111

9. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
    printf("\n");
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2  
☐ 0 1 2 0 1 2  
☐ 1 2 3 1 2  
☐ 0 1 0 1 0 1

10. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7  
☐ 0  
☐ 5

11. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases  
☐ ses champs  
☐ ses chants  
☐ ses blocs

12. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5

13. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 5
- ☐ 4
- ☐ 3

14. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 2
- ☐ 8 6 4 2 0
- ☐ 8 6 4 2
- ☐ 0 2 4 6 8

15. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1
- ☐ 0 1 2 3
- ☐ 4 3 2 1 0

16. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`

17. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max' , que doit-on chercher dans le programme ?**

- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction

- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration

18. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur

19. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`
- ☐ `void exposant(double x^n);`
- ☐ `double exposant(double x, int n);`
- ☐ `int exposant(double n, int x);`

20. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable x vaut 0
- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 5 et la variable y vaut 0

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Quels calculs peut-on programmer en programmation structurée ?

- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine

2. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1

3. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`
- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(double n);`

4. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

5. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

6. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

7. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char c;`
- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `int char;`

8. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ rien

9. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define taille = 3`
- ☐ `#define N = 3`
- ☐ `#define N 3`
- ☐ `#define taille = N`

10. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(struct date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

11. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un répertoire
- ☐ changer de répertoire courant
- ☐ ouvrir un fichier texte
- ☐ créer un fichier texte

12. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b
- ☐ A
- ☐ C
- ☐ B

13. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\* \*\*
- ☐ \*\* \*\* \*\* \*\*
- ☐ \*\* \*\*\* \*\*
- ☐ \*\*\*\* \*\*

14. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale
- ☐ analyse harmonique
- ☐ analyse syntaxique
- ☐ analyse sémantique

15. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 0
- ☐ 4
- ☐ 1
- ☐ 5

16. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x - 1 = racine(x);`

17. Un fichier source est :

- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document qui doit être protégé
- ☐ un document de référence du système
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document illisible pour les humains

18. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 7
- ☐ 111
- ☐ 8

19. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1
- ☐ 1 2 3 1 2
- ☐ 0 0 1 1 2 2
- ☐ 0 1 2 0 1 2

20. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ la variable `y` vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable `x` vaut 5 et la variable `y` vaut 0
- ☐ la variable `x` vaut 0



**Barème : 1 points par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`  
☐ `x = racine(2/3);`  
☐ `x = racine(racine(x)*racine(x));`  
☐ `x - 1 = racine(x);`

2. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur  
☐ Majeur  
☐ Majeur  
☐ Mineur  
☐ rien

3. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction  
☐ alphabétique  
☐ un ordre quelconque  
☐ dans lequel ces fonctions sont appelées dans le main

4. Sous unix (ou linux), la commande **cd** permet de :

- ☐ ouvrir un bureau partagé (common desktop)  
☐ jouer de la musique  
☐ détruire un fichier  
☐ récupérer un programme arrêté avec la commande **ab**  
☐ changer de répertoire courant

5. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse sémantique  
☐ analyse lexicale  
☐ analyse syntaxique  
☐ analyse harmonique

6. Pour déclarer une fonction **pgcd** qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`  
☐ `int pgcd(int y, int x);`  
☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, int x);`

7. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 4  
☐ j = 0  
☐ j = %d  
☐ j = 5

8. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »**

- ☐ l'édition de liens  
☐ l'analyse des entrées clavier  
☐ l'analyse sémantique  
☐ l'analyse harmonique

9. Quels calculs peut-on programmer en programmation structurée ?

- ☐ certains programmes sont de vrais plats de spaghetti  
☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine  
☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine  
☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée

10. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ des processus  
☐ en temps d'accès  
☐ les fichiers du disque  
☐ certaines données de la mémoire de travail

11. Soit la fonction **g** définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression **g(0)** prendra la valeur :

- ☐ 7  
☐ 5  
☐ 0

12. Si cette erreur apparaît à la compilation :

**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade manquante  
☐ un point-virgule en trop  
☐ une accolade en trop  
☐ un point-virgule manquant

13. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il comporte une boucle infinie

14. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int[] nouveau(5);`
- ☐ `char tableau[5];`
- ☐ `int tab[] = 5;`
- ☐ `int toto[taille=5];`
- ☐ `int toto[5];`

15. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`  
, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction appelée avant sa déclaration

16. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 3
- ☐ 5
- ☐ 4

17. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ `!(A || B) == (A && !B)`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `A && B`
- ☐ `(!A || B)`

18. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ Arriver à l'heure en cours
- ☐ Transférer des données et instructions entre processeur et mémoire
- ☐ transporter les processus du tourniquet au processeur

19. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que n est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`
- ☐ `int n = carre();`
- ☐ `int carre(2);`

20. Si x est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

- ☐ 1
- ☐ 0
- ☐ 1.5
- ☐ 0.5

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

☐ `int factorielle(int 2);`  
☐ `printf("%d", factorielle(n));`  
☐ `n = factorielle();`  
☐ `n = factorielle(p, q);`

2. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

☐ 0.5  
☐ 0  
☐ 1  
☐ 1.5

3. Une *segmentation fault* est une erreur qui survient lorsque :

☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur  
☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée  
☐ la division du programme en zones homogènes échoue  
☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal

4. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

☐ Mineur  
☐ Majeur  
☐ Mineur Majeur  
☐ rien

5. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

☐ 0 1 2 3 4  
☐ 4 3 2 1 0  
☐ 0 1 2 3  
☐ 4 3 2 1

6. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

☐ tous ensemble  
☐ chacun son tour, après que le processus précédent a terminé  
☐ en parallèle, chacun dans un registre  
☐ tour à tour, un petit peu à chaque fois

7. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

☐ `f(a,b)=13, a=8, b=5`  
☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=8, a=3, b=5`  
☐ `f(3,5)=8, a=3, b=5`

8. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

☐ \*\*\*\*\*  
☐ \*\* \*\* \*\* \*\*  
☐ \*\* \*\* \*\* \*\*  
☐ \*\*\*\* \*\* \*\* \*

9. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

☐ sélectionner entre deux blocs à l'aide d'une condition  
☐ mettre les blocs en séquence les uns à la suite des autres  
☐ répéter un bloc tant qu'une condition est vérifiée  
☐ retourner un bloc

10. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

☐ `for(i=1;i<5;i=i+1)`  
☐ `for(i=0;i<5;i=i+1)`  
☐ `for(i=0;i<=5;i=i+1)`  
☐ `for(i=1;i<=5;i=i+1)`

11. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses chants
  - ☐ ses champs
  - ☐ ses cases
  - ☐ ses blocs
12. Vous utilisez une boucle **while** quand :
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous avez déjà fait un **for** dans le même programme principal
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ l'incrément de la variable de boucle n'est pas 1
13. Au début de la fonction **main()** on place le code :
- ```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```
- Alors l'affichage sera :
- ☐ ABCDEF
  - ☐ A
  - ☐ cccccc
  - ☐ i
14. Dans la commande gcc, l'option **-Wall** signifie :
- ☐ qu'on veut changer aléatoirement de fond d'écran
  - ☐ qu'il faut lancer un débogueur
  - ☐ que l'on veut voir tous les avertissements
  - ☐ qu'il faut indenter le fichier source

15. L'écriture 111 en binaire correspond au nombre naturel :
- ☐ 7
  - ☐ 111
  - ☐ 3
  - ☐ 8
16. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
- ☐ **char "c";**
  - ☐ **char 'c';**
  - ☐ **int char;**
  - ☐ **char c;**
17. Si *n* est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
- ☐ un débogueur
  - ☐ **scanf("%d", &n);**
  - ☐ **printf("Valeur de n ? %d\n", n);**
  - ☐ **printf("Valeur de n ? %g\n", n);**
18. Soit un programme contenant les lignes suivantes :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ **j = %d**
- ☐ **j = 0**
- ☐ **j = 4**
- ☐ **j = 5**

19. Pour déclarer une procédure **afficher\_date** qui prend en argument un **struct date\_s** et affiche le contenu du struct, on écrit :
- ☐ **void afficher\_date(date\_s d);**
  - ☐ **struct date\_s afficher\_date(struct date\_s d);**
  - ☐ **void afficher\_date(struct date\_s d);**
  - ☐ **int afficher\_date(date\_s d);**

20. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 4
- ☐ 1
- ☐ 0
- ☐ 5

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 4
- ☐ j = 0
- ☐ j = 5
- ☐ j = %d

2. Vous utilisez une boucle **while** quand :

- ☐ vous avez déjà fait un **for** dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1

3. Un bit est :

- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur
- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme

4. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur
- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `printf("Valeur de n ? %g\n", n);`

5. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ certaines données de la mémoire de travail
- ☐ les fichiers du disque
- ☐ des processus
- ☐ en temps d'accès

6. Le bus système sert à :

- ☐ transporter les processus du tourniquet au processeur
- ☐ Écrire des données sur le disque dur
- ☐ Transférer des données et instructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours

7. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse sémantique
- ☐ analyse syntaxique
- ☐ analyse harmonique
- ☐ analyse lexicale

8. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y = 3;
13
14     x = y;
15
16     ...
17 }
```

- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 3
- ☐ la variable y vaut 5
- ☐ la variable x vaut 5 et la variable y vaut 3

9. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(a<n) || (n>b)`
- ☐ `a<=n<=b`

10. Si **carre** est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que  $n$  est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int n = carre();`
- ☐ `n = carre(int n);`
- ☐ `int carre(2);`
- ☐ `n = carre(n);`

11. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char 'c';`

12. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1

13. Si cette erreur apparaît à la compilation :  
**error: expected ‘;’ before ‘}’ token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop
- ☐ un point-virgule manquant
- ☐ une accolade manquante
- ☐ une accolade en trop

14. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses chants
- ☐ ses cases
- ☐ ses champs

15. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Majeur
- ☐ Mineur
- ☐ Mineur  
Majeur

16. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine

17. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for ‘max’**, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d’une fonction
- ☐ une directive préprocesseur **#include** manquante
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration

18. L’écriture 101 en binaire correspond au nombre naturel :

- ☐ 4
- ☐ 101
- ☐ 3
- ☐ 5

19. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l’expression **f(0)** prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

20. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`
- ☐ `double exposant(double x, int n);`
- ☐ `int exposant(double n, int x);`
- ☐ `exposant(double x, int n, int r);`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un fichier source est :

- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document qui doit être protégé
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un document illisible pour les humains
- ☐ un document de référence du système

2. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine

3. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut indenter le fichier source
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut lancer un débogueur

4. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int y, int x);`
- ☐ `void pgcd(int x, int y);`

5. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <stdio.h>`
- ☐ `#include <studio.h>`

6. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

7. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(struct date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

8. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ certaines données de la mémoire de travail
- ☐ des processus
- ☐ les fichiers du disque
- ☐ en temps d'accès

9. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui ne sera pas détectée
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ ne comportent aucune erreur

10. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Mineur
- ☐ Mineur Majeur
- ☐ Majeur

11. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1

12. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :
- ☐ `n = carre(n);`
  - ☐ `n = carre(int n);`
  - ☐ `int n = carre();`
  - ☐ `int carre(2);`
13. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :
- ☐ `printf("x=%x et y=%y\n");`
  - ☐ `printf("x=%d et y=%d\n",x,y);`
  - ☐ `printf("x=%d et y=%d\n,x,y");`
  - ☐ `printf("x=%d et y=%d\n",x y);`
14. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses blocs
  - ☐ ses chants
  - ☐ ses cases
  - ☐ ses champs
15. Une *segmentation fault* est une erreur qui survient lorsque :
- ☐ la division du programme en zones homogènes échoue
  - ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
  - ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
16. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?
- ☐ `char "c";`
  - ☐ `char 'c';`
  - ☐ `char c;`
  - ☐ `int char;`
17. Si le code :
- ```
struct toto_s
{
    int n;
    double x;
};
```
- précède la fonction `main()`, alors on peut écrire en début de `main()` :
- ☐ `int toto.n = 3;`
  - ☐ `struct toto_s toto;`
  - ☐ `int struct toto_s = {3, -1e10};`
  - ☐ `toto_s n, x;`
  - ☐ `toto_s struct z = {3, 0.5};`

18. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
  - ☐ 3
  - ☐ 4
19. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :
- ☐ 0.5
  - ☐ 1.5
  - ☐ 1
  - ☐ 0
20. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?
- ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction appelée avant sa déclaration
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction déclarée mais non définie



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1 0 1  
☐ 0 1 2 0 1 2  
☐ 1 2 1 2 3  
☐ 0 0 0 1 1 1

2. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4  
☐ 4 3 2 1 0  
☐ 1 2 3 4  
☐ 4 3 2 1

3. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`  
☐ `struct int factorielle(int n);`  
☐ `int factorielle(double n);`  
☐ `int factorielle();`

4. Soit la fonction **g** définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression **g(0)** prendra la valeur :

- ☐ 5  
☐ 0  
☐ 7

5. Si **a** et **b** sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de **a** et de **b** on utilise la condition :

- ☐ `a = b`  
☐ `(a.n == b.n) && (a.x == b.x)`  
☐ `a{n, x} == b{n, x}`  
☐ `a == b`

6. Vous utilisez une boucle **while** quand :

- ☐ vous avez déjà fait un **for** dans le même programme principal  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous n'avez pas déclaré de fonction

7. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0; i<=5; i=i+1)`  
☐ `for(i=1; i<5; i=i+1)`  
☐ `for(i=0; i<5; i=i+1)`  
☐ `for(i=1; i<=5; i=i+1)`

8. Après exécution du programme :

```
1 lecture 8 r0
2 valeur 3 r1
3 mult r1 r0
4 valeur 1 r2
5 add r2 r0
6 ecriture r0 8
7 stop
8 5
```

- ☐ la case mémoire 8 contiendra 16  
☐ le bus explose  
☐ le terminal affiche 8  
☐ la case mémoire 8 contiendra 0

9. L'ordonnancement par tourniquet permet :

- ☐ de ne pas perdre de temps avec la commutation de contexte  
☐ d'afficher des ronds colorés à l'écran  
☐ de doubler la mémoire disponible  
☐ d'entretenir l'illusion que les processus tournent en parallèle

10. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `x = mccarthy(n);`  
☐ `n = mccarthy();`  
☐ `n = mccarthy(p, q);`  
☐ `int mccarthy(int 2);`

11. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4  
☐ 4 3 2 1  
☐ 0 1 2 3  
☐ 4 3 2 1 0

12. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A
- ☐ cccccc
- ☐ ABCDEF
- ☐ i

13. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `int char;`
- ☐ `char 'c';`
- ☐ `char c;`
- ☐ `char "c";`

14. Le langage C est un langage

- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ interprété
- ☐ composé

15. Quel est le problème d'un programme comportant les lignes suivantes?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou

16. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3
- ☐ 6
- ☐ 20
- ☐ 16

17. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int tab[] = 5;`

- ☐ `char tableau[5];`
- ☐ `int[] new tableau(5);`
- ☐ `int toto[taille=5];`
- ☐ `int toto[5];`

18. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `kwrite TP4`
- ☐ `yppasswd`
- ☐ `new TP4`
- ☐ `mkdir TP4`

19. Le type des réels en C est :

- ☐ `int`
- ☐ `char`
- ☐ `real`
- ☐ `double`

20. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, y);`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 4
- ☐ 101
- ☐ 5

2. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int[] new tableau(5);`
- ☐ `int tab[] = 5;`
- ☐ `int toto[taille=5];`
- ☐ `char tableau[5];`
- ☐ `int toto[5];`

3. Quels calculs peut-on programmer en programmation structurée ?

- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée

4. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1

5. Le type des réels en C est :

- ☐ `double`
- ☐ `real`
- ☐ `char`
- ☐ `int`

6. Au début de la fonction `main()` on place le code :

```
char b = 'A';  
b = b + 2;  
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A
- ☐ C
- ☐ b
- ☐ B

7. Le bus système sert à :

- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ transporter les processus du tourniquet au processeur

8. Pour l'extrait de programme suivant :

```
int produit = 0;  
int serie[4] = {2, 2, 2, 2};  
for (i = 0; i < 4; i = i + 1)  
{  
    produit = produit * serie[i];  
}  
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 0
- ☐ 4
- ☐ 8

9. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle(p, q);`
- ☐ `n = factorielle();`
- ☐ `int factorielle(int 2);`
- ☐ `printf("%d", factorielle(n));`

10. Pour l'extrait de programme suivant :

```
int i = 0;  
int j = 0;  
for (i = 0; i < 2; i = i + 1)  
{  
    for (j = 0; j < 3; j = j + 1)  
    {  
        printf("%d ", j);  
    }  
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 3 0 1 2

11. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char c;`

12. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 1.5
- ☐ 0
- ☐ 1
- ☐ 0.5

13. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 4 3 2 1

14. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int y, int x);`

15. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 1 2 3 4

16. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`

17. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`

- ☐ `(a<n) || (n>b)`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`

18. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant
- ☐ une accolade manquante
- ☐ un point-virgule en trop
- ☐ une accolade en trop

19. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`
- ☐ `int factorielle(double n);`
- ☐ `struct int factorielle(int n);`

20. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%d et y=%d\n",x y);`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*
- ☐ \*\*\*\*\*
- ☐ \*\*\*\*
- ☐ \*\* \*\*

2. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur
- ☐ Mineur  
Majeur
- ☐ rien
- ☐ Majeur

3. Le langage C est un langage

- ☐ composé
- ☐ lu, écrit, parlé
- ☐ interprété
- ☐ compilé

4. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`
- ☐ `(a<=n) && (n<=b)`
- ☐ `a<=n<=b`
- ☐ `(n<=a) && (n<=b)`

5. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur `#include` manquante

6. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur
- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `printf("Valeur de n ? %d\n", n);`

7. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `double afficher_menu();`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `char afficher_menu(printf("menu"));`

8. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas

9. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

10. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses chants
- ☐ ses blocs
- ☐ ses cases

11. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studlib.h>`
- ☐ `#include <studio.h>`

12. Une variable booléenne est une variable :

- ☐ réelle positive
- ☐ qui est vraie ou fausse
- ☐ à laquelle une valeur vient d'être affectée
- ☐ jamais nulle
- ☐ NaN (not a number, qui n'est pas un nombre)

13. Quel est l'opérateur de différence en C :

- ☐ <>
- ☐ !=
- ☐ !
- ☐ ≠

14. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4

15. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
```

```
15
16      ...
17  }
```

- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 5 et la variable y vaut 3
- ☐ la variable x vaut 3

16. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction

17. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tous ensemble
- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ tour à tour, un petit peu à chaque fois
- ☐ en parallèle, chacun dans un registre

18. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(int n);`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `int saisie_utilisateur();`

19. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ `loop i;`
- ☐ `int %d;`
- ☐ `int loop n;`
- ☐ `int k;`

20. Un fichier source est :

- ☐ un document de référence du système
- ☐ un document illisible pour les humains
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document qui doit être protégé
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique
- ☐ un ordre quelconque
- ☐ dans lequel ces fonctions sont appelées dans le main

2. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante

3. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char c;`

4. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`
- ☐ `void exposant(double x^n);`
- ☐ `exposant(double x, int n, int r);`

5. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
```

```
{
    return f(a - 1) + 1;
}
return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 0
- ☐ 1
- ☐ 5
- ☐ 4

6. Soit la fonction **g** définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression **g(0)** prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 7

7. Si **pgcd** est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `int n = pgcd();`
- ☐ `int pgcd(2);`

8. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42
- ☐ 1
- ☐ 6
- ☐ 0

9. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

**Undefined symbols : "\_printf" ou référence indéfinie vers < printf >**

- ☐ l'analyse harmonique
- ☐ l'analyse sémantique
- ☐ l'analyse des entrées clavier
- ☐ l'édition de liens

10. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n", f(a,b), a, b);
    return EXIT_SUCCESS;
}
```

appelant la fonction **f** ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

11. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
```

- ```
15
16    ...
17 }
```
- ☐ la variable y vaut 5
- ☐ la variable x vaut 5 et la variable y vaut 0
- ☐ la variable x vaut 0
- ☐ le programme affiche "Faux"
12. Le code suivant :
- ```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 8 2
- ☐ 8 6 4 2 0
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2
13. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ un débogueur
- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %d\n", n);`
14. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :
- ☐ `double afficher_menu();`
- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu(int char);`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu();`

15. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :
- ☐ analyse sémantique
- ☐ analyse harmonique
- ☐ analyse lexicale
- ☐ analyse syntaxique
16. Un fichier source est :
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document illisible pour les humains
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document de référence du système
- ☐ un document qui doit être protégé
17. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17 }
```
- ☐ la variable y vaut 5
- ☐ la variable x vaut 5 et la variable y vaut 3
- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 3
18. Au début de la fonction `main()` on place le code :
- ```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

- Alors l'affichage sera :
- ☐ i
- ☐ cccccc
- ☐ A
- ☐ ABCDEF
19. Pour l'extrait de programme suivant :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```
- qu'est ce qui sera affiché ?
- ☐ 1 2 3 1 2
- ☐ 0 0 1 1 2 2
- ☐ 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
20. Le code suivant :
- ```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 0 1 2 3



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un registre du processeur est :

- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

2. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`

3. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`

4. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2
- ☐ 8 2
- ☐ 8 6 4 2 0
- ☐ 0 2 4 6 8

5. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses champs
- ☐ ses cases
- ☐ ses blocs

6. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int y, int x);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`

7. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ en temps d'accès
- ☐ les fichiers du disque
- ☐ des processus
- ☐ certaines données de la mémoire de travail

8. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 7
- ☐ 111
- ☐ 8

9. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`
- ☐ `a<=n<=b`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<=n) && (n<=b)`

10. Quel est l'opérateur de différence en C :

- ☐ `≠`
- ☐ `!`
- ☐ `<>`
- ☐ `!=`

11. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas

12. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un répertoire
- ☐ créer un fichier texte
- ☐ changer de répertoire courant
- ☐ ouvrir un fichier texte

13. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `!(A || B) == (A && !B)`
- ☐ `(!A || B)`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `A && B`

14. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

☐ `void saisie_utilisateur(int n);`  
☐ `void saisie_utilisateur(char c);`  
☐ `int saisie_utilisateur();`  
☐ `saisie_utilisateur scanf(%d);`

15. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

☐ un ordre quelconque  
☐ dans lequel vous avez déclaré ces fonction  
☐ alphabétique  
☐ dans lequel ces fonctions sont appelées dans le `main`

16. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

☐ 6  
☐ 16  
☐ 3  
☐ 20

17. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x  = %d\n", 2);
14
15      ...
16  }
```

☐ le terminal affiche 5  
☐ le terminal affiche "Faux"  
☐ le terminal affiche `x = 2`  
☐ le terminal affiche `x = 5`

18. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
```

```
}
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

☐ 7  
☐ 5  
☐ 0

19. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

☐ `x = racine(racine(x)*racine(x));`  
☐ `x = racine(x * x) - racine(x);`  
☐ `x - 1 = racine(x);`  
☐ `x = racine(2/3);`

20. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

☐ `int n = carre();`  
☐ `n = carre(n);`  
☐ `n = carre(int n);`  
☐ `int carre(2);`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction appelée avant sa déclaration
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction déclarée mais non définie
- Sous unix (ou linux), la commande `cd` permet de :
  - ☐ jouer de la musique
  - ☐ détruire un fichier
  - ☐ changer de répertoire courant
  - ☐ ouvrir un bureau partagé (common desktop)
  - ☐ récupérer un programme arrêté avec la commande `ab`
- Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :
  - ☐ en parallèle, chacun dans un registre
  - ☐ chacun son tour, après que le processus précédent a terminé
  - ☐ tour à tour, un petit peu à chaque fois
  - ☐ tous ensemble
- Pour l'extrait de programme suivant :
 

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

 La valeur de somme affichée est :
  - ☐ 3
  - ☐ 6
  - ☐ 16
  - ☐ 20

- Le code suivant :
 

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

 affichera :
  - ☐ 4 3 2 1
  - ☐ 0 1 2 3 4
  - ☐ 1 2 3 4
  - ☐ 4 3 2 1 0
- Vous utilisez une boucle `while` quand :
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous avez déjà fait un `for` dans le même programme principal
- Soit la fonction `f` définie par :
 

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

 Alors l'expression `f(0)` prendra la valeur :
  - ☐ 0
  - ☐ 3
  - ☐ 4
- On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?
  - ☐ `(!A || B)`
  - ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `A && B`
  - ☐ `!(A || B) == (A && !B)`

- Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
  - ☐ `double exposant(double x, int n);`
  - ☐ `exposant(double x, int n, int r);`
  - ☐ `int exposant(double n, int x);`
  - ☐ `void exposant(double x^n);`
- Au début de la fonction `main()` on place le code :
 

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

 Alors l'affichage sera :
  - ☐ i
  - ☐ ABCDEF
  - ☐ cccccc
  - ☐ A
- Pour l'extrait de programme suivant :
 

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

 La valeur affichée est :
  - ☐ 0
  - ☐ 16
  - ☐ 8
  - ☐ 4
- Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
  - ☐ `char c;`
  - ☐ `char "c";`
  - ☐ `char 'c';`
  - ☐ `int char;`

13. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\* \*\*
- ☐ \*\* \*\* \*\* \*\*
- ☐ \*\*\*\* \*\*
- ☐ \*\* \*\*

14. Sous unix (ou linux), la commande **ls** permet de :

- ☐ voir des clips musicaux
- ☐ afficher le contenu d'un fichier texte
- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ compiler un programme

15. Un registre du processeur est :

- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une unité de calcul spécialisée de l'ordinateur

16. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou

17. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses chants
- ☐ ses cases
- ☐ ses blocs

18. Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :

- ☐ `double afficher_menu();`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu();`
- ☐ `char afficher_menu(sprintf("menu"));`
- ☐ `int afficher_menu(int char);`

19. Un fichier source est :

- ☐ un document qui doit être protégé
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document illisible pour les humains
- ☐ un document de référence du système

20. Lorsqu'un programme utilise **printf** ou **scanf** il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <stdio.h>`
- ☐ `#include <studio.h>`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 0
- ☐ 42
- ☐ 1
- ☐ 6

2. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses champs
- ☐ ses cases
- ☐ ses blocs

3. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade manquante
- ☐ un point-virgule en trop
- ☐ une accolade en trop
- ☐ un point-virgule manquant

4. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s n, x;`
- ☐ `struct toto_s toto;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `int toto.n = 3;`

5. Quel est l'opérateur de différence en C :

- ☐ !
- ☐ <>
- ☐ !=
- ☐ ≠

6. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

- ☐ 0.5
- ☐ 1
- ☐ 0
- ☐ 1.5

7. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(int n);`
- ☐ `int n = carre();`
- ☐ `n = carre(n);`
- ☐ `int carre(2);`

8. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ Arriver à l'heure en cours
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ transporter les processus du tourniquet au processeur

9. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1

10. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 0 1 2 3 4

11. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

12. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8

- ☐ 16
- ☐ 4
- ☐ 0

13. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 0
- ☐ 3

14. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 1
- ☐ 4
- ☐ 5
- ☐ 0

15. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char c;`
- ☐ `char "c";`
- ☐ `char 'c';`

16. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ A
- ☐ cccccc
- ☐ i

17. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 1 2 1 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 0 1 0 1 0 1
- ☐ 0 0 0 1 1 1

18. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ `int %d;`
- ☐ `int k;`
- ☐ `int loop n;`
- ☐ `loop i;`

19. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration

20. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 16
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

☐ `void afficher_date(date_s d);`  
☐ `int afficher_date(date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`  
☐ `void afficher_date(struct date_s d);`

2. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

☐ `int factorielle();`  
☐ `int factorielle(double n);`  
☐ `struct int factorielle(int n);`  
☐ `int factorielle(int x);`

3. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, y);`  
☐ `int pgcd(int x, int x);`  
☐ `int pgcd(int y, int x);`

4. Vous utilisez une boucle `while` quand :

☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance  
☐ vous avez déjà fait un `for` dans le même programme principal  
☐ vous n'avez pas déclaré de fonction

5. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

☐ rien  
☐ Mineur  
☐ Majeur  
☐ Mineur  
☐ Majeur

6. Sur unix (ou linux), la commande `mkdir` permet de :

☐ créer un fichier texte  
☐ changer de répertoire courant  
☐ ouvrir un fichier texte  
☐ créer un répertoire

7. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

☐ `int n = carre();`  
☐ `n = carre(n);`  
☐ `n = carre(int n);`  
☐ `int carre(2);`

8. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

☐ `char 'c';`  
☐ `char c;`  
☐ `int char;`  
☐ `char "c";`

9. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

☐ `j = 0`  
☐ `j = 5`  
☐ `j = %d`  
☐ `j = 4`

10. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

☐ 10  
☐ 15  
☐ 6  
☐ 0

11. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

☐ `for(i=1;i<=5;i=i+1)`  
☐ `for(i=0;i<=5;i=i+1)`  
☐ `for(i=1;i<5;i=i+1)`  
☐ `for(i=0;i<5;i=i+1)`

12. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

☐ `int afficher_menu(int char);`  
☐ `int afficher_menu();`  
☐ `double afficher_menu();`  
☐ `char afficher_menu(printf("menu"));`  
☐ `void afficher_menu();`

13. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

☐ `printf("Valeur de n ? %d\n", n);`  
☐ un débogueur  
☐ `printf("Valeur de n ? %g\n", n);`  
☐ `scanf("%d", &n);`

14. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 0
- ☐ 4
- ☐ 8
- ☐ 16

15. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 3 0 1 2
- ☐ 0 0 1 1 2 2 3

16. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse lexicale
- ☐ analyse syntaxique
- ☐ analyse sémantique

17. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

18. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
```

```
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 20
- ☐ 16
- ☐ 3

19. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 7
- ☐ 8
- ☐ 3
- ☐ 111

20. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- Si cette erreur apparaît à la compilation :  
`error: expected ';' before '}' token` que doit-on chercher dans le programme ?
  - ☐ une accolade en trop
  - ☐ un point-virgule manquant
  - ☐ un point-virgule en trop
  - ☐ une accolade manquante
- Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :
  - ☐ `n = pgcd(int p, int q);`
  - ☐ `n = pgcd(n, 3);`
  - ☐ `int n = pgcd();`
  - ☐ `int pgcd(2);`
- Le code suivant :
 

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

 affichera :
  - ☐ 4 3 2 1 0
  - ☐ 4 3 2 1
  - ☐ 1 2 3 4
  - ☐ 0 1 2 3 4
- Après exécution jusqu'à la ligne 15 du programme C :
 

```
10 ...
11 int main() {
12     int x = 5;
13
14     x = 3 * x + 1;
15
16     ...
17 }
```

  - ☐ la variable `x` vaut 16
  - ☐ le programme affiche `****`
  - ☐ la variable `x` vaut  $-\frac{1}{2}$
  - ☐ le programme affiche `x`

- Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
  - ☐ 7
  - ☐ 5
- Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
    - ☐ `#appart <stdlib.h>`
    - ☐ `#include <studlib.h>`
    - ☐ `#include <stdio.h>`
    - ☐ `#include <studio.h>`
  - L'écriture 101 en binaire correspond au nombre naturel :
    - ☐ 5
    - ☐ 101
    - ☐ 3
    - ☐ 4
  - Si le code :
 

```
struct toto_s
{
    int n;
    double x;
};
```

 précède la fonction `main()`, alors on peut écrire en début de `main()` :
    - ☐ `int toto.n = 3;`
    - ☐ `toto_s n, x;`
    - ☐ `struct toto_s toto;`
    - ☐ `int struct toto_s = {3, -1e10};`
    - ☐ `toto_s struct z = {3, 0.5};`

- On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `!(A || B) == (A && !B)`
- ☐ `(!A || B)`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `A && B`

- Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur
- ☐ qu'il faut indenter le fichier source

- Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ la variable `y` vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable `x` vaut 5 et la variable `y` vaut 0
- ☐ la variable `x` vaut 0

- Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy();`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy(p, q);`
- ☐ `x = mccarthy(n);`

13. Sur unix (ou linux), la commande `mkdir` permet de :
- ☐ ouvrir un fichier texte
  - ☐ créer un répertoire
  - ☐ changer de répertoire courant
  - ☐ créer un fichier texte
14. Soit la fonction `f` définie par :
- ```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```
- Alors l'expression `f(0)` prendra la valeur :
- ☐ 3
  - ☐ 0
  - ☐ 4
15. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`  
, que doit-on chercher dans le programme ?
- ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction déclarée mais non définie
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction appelée avant sa déclaration
16. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :
- ☐ en temps d'accès

- ☐ certaines données de la mémoire de travail
  - ☐ les fichiers du disque
  - ☐ des processus
17. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```
- ☐ le programme affiche "Faux"
  - ☐ la variable `y` vaut 5
  - ☐ la variable `x` vaut 5 et la variable `y` vaut 3
  - ☐ la variable `x` vaut 3
18. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses chants
  - ☐ ses cases
  - ☐ ses blocs
  - ☐ ses champs
19. Le code suivant :
- ```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
  - ☐ Mineur
  - ☐ rien
  - ☐ Mineur  
Majeur
20. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :
- ```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```
- Quelle est la condition `cond` :
- ☐ `(a<n) || (n>b)`
  - ☐ `(a<=n) && (n<=b)`
  - ☐ `(n<=a) && (n<=b)`
  - ☐ `a<=n<=b`

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un registre du processeur est :

- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ une gamme de fréquence de fonctionnement du processeur

2. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6
- ☐ 0 2 4 6 8
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6

3. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 5
- ☐ 3
- ☐ 101
- ☐ 4

4. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 0 1 2 3
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4

5. Le langage C est un langage

- ☐ compilé
- ☐ composé
- ☐ interprété
- ☐ lu, écrit, parlé

6. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(2/3);`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`

7. Pour déclarer une fonction **pgcd** qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int y, int x);`

8. Au début de la fonction **main()** on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ A
- ☐ i
- ☐ cccccc

9. Si cette erreur apparaît à la compilation :

Undefined symbols : "\_printf" ou référence indéfinie vers « printf » que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ un caractère interdit en C
- ☐ une variable non déclarée
- ☐ une faute de frappe dans un appel de fonction

10. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 1 2 3 1 2
- ☐ 0 1 0 1 0 1
- ☐ 0 0 1 1 2 2

11. Si cet avertissement apparaît à la compilation :

**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction

12. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 3 0 1 2
- ☐ 0 1 2 0 1 2 3

13. Si  $x$  est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1.5
- ☐ 0.5
- ☐ 0
- ☐ 1

14. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses blocs
- ☐ ses chants
- ☐ ses champs

15. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant
- ☐ un point-virgule en trop
- ☐ une accolade en trop
- ☐ une accolade manquante

16. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
```

```
14      x = y;
15
16      ...
17  }
```

- ☐ la variable  $x$  vaut 3
- ☐ la variable  $x$  vaut 5 et la variable  $y$  vaut 3
- ☐ le programme affiche "Faux"
- ☐ la variable  $y$  vaut 5

17. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction  $f$  ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐  $f(a,b)=8$ ,  $a=8$ ,  $b=5$
- ☐  $f(a,b)=8$ ,  $a=3$ ,  $b=5$

- ☐  $f(3,5)=8$ ,  $a=3$ ,  $b=5$
- ☐  $f(a,b)=13$ ,  $a=8$ ,  $b=5$

18. Vous utilisez une boucle **while** quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un **for** dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction

19. Pour compiler un programme **prog.c**, on utilise la ligne de commande :

- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.exe -Wall -o prog.c`

20. Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu(int char);`
- ☐ `char afficher_menu(sprintf("menu"));`
- ☐ `int afficher_menu();`
- ☐ `double afficher_menu();`
- ☐ `void afficher_menu();`

**Barème : 1 point par réponse juste (unique) ; -0,5 point par réponse fausse. Durée : 20 minutes.**

1. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque

2. Le type des réels en C est :

- ☐ double
- ☐ int
- ☐ char
- ☐ real

3. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ des processus
- ☐ les fichiers du disque
- ☐ certaines données de la mémoire de travail
- ☐ en temps d'accès

4. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `int exposant(double n, int x);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `double exposant(double x, int n);`
- ☐ `void exposant(double x^n);`

5. Le code suivant :

```
int i;
for (i = 4; i > 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3
- ☐ 0 1 2 3 4

6. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ `a = b`
- ☐ `a == b`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a{n, x} == b{n, x}`

7. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 5 et la variable y vaut 0
- ☐ la variable x vaut 0
- ☐ la variable y vaut 5

8. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou

9. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ `** *** **** *****`
- ☐ `**** ***** **** *`
- ☐ `** ** ** ** **`
- ☐ `***** ***** *** **`

10. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(2/3);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(x * x) - racine(x);`

11. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `char "c";`
- ☐ `char c;`
- ☐ `int char;`

12. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 4
- ☐ j = 0
- ☐ j = %d
- ☐ j = 5

13. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

14. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 0 1 2 3 4

15. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`

16. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%d et y=%d\n",x,y);`

17. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`

18. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme?

- ☐ un point-virgule manquant
- ☐ une accolade en trop
- ☐ un point-virgule en trop
- ☐ une accolade manquante

19. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse syntaxique
- ☐ analyse sémantique
- ☐ analyse lexicale
- ☐ analyse harmonique

20. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A
- ☐ b
- ☐ B
- ☐ C

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

- Quels calculs peut-on programmer en programmation structurée ?
  - ☐ certains programmes sont de vrais plats de spaghetti
  - ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
  - ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
  - ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- Avant de faire appel à une fonction il est nécessaire de :
  - ☐ l'avoir définie
  - ☐ l'avoir déclarée et définie
  - ☐ l'avoir déclarée
  - ☐ avoir défini une constante symbolique de la taille de cette fonction

- Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5

- Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
  - ☐ `scanf("%d", &n);`
  - ☐ un débogueur
  - ☐ `printf("Valeur de n ? %g\n", n);`
  - ☐ `printf("Valeur de n ? %d\n", n);`
- Le langage C est un langage
  - ☐ interprété
  - ☐ composé
  - ☐ compilé
  - ☐ lu, écrit, parlé
- Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.
  - ☐ `#define taille = 3`
  - ☐ `#define N 3`
  - ☐ `#define taille = N`
  - ☐ `#define N = 3`
- Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :
  - ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x - 1 = racine(x);`
  - ☐ `x = racine(2/3);`
  - ☐ `x = racine(x * x) - racine(x);`
- Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
  - ☐ `#include <studio.h>`
  - ☐ `#include <studlib.h>`
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <stdio.h>`
- Un bit est :
  - ☐ l'instruction qui met fin à un programme
  - ☐ un battement d'horloge processeur
  - ☐ un chiffre binaire (0 ou 1)
  - ☐ la longueur d'un mot mémoire

- Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a{n, x} == b{n, x}`
- ☐ `a == b`
- ☐ `a = b`
- ☐ `(a.n == b.n) && (a.x == b.x)`

- On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`

- Un fichier source est :

- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un document qui doit être protégé
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document illisible pour les humains
- ☐ un document de référence du système

13. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 0
- ☐ la variable x vaut 5 et la variable y vaut 0
- ☐ la variable y vaut 5

14. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`

15. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`

16. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ alphabétique

17. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `struct toto_s toto;`
- ☐ `int toto.n = 3;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `toto_s n, x;`

18. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

Undefined symbols : "\_printf" ou  
référence indéfinie vers « printf »

- ☐ l'analyse harmonique
- ☐ l'analyse sémantique
- ☐ l'analyse des entrées clavier
- ☐ l'édition de liens

19. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 5
- ☐ 101
- ☐ 4

20. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0



**Barème : 1 points par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
  - ☐ `exposant(double x, int n, int r);`
  - ☐ `int exposant(double n, int x);`
  - ☐ `double exposant(double x, int n);`
  - ☐ `void exposant(double x^n);`
- Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
  - ☐ `#include <studio.h>`
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <studlib.h>`
  - ☐ `#include <stdio.h>`
- Vous utilisez une boucle `while` quand :
  - ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ vous n'avez pas déclaré de fonction
- Le type des réels en C est :
  - ☐ `int`
  - ☐ `double`
  - ☐ `real`
  - ☐ `char`
- Au début de la fonction `main()` on place le code :
 

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

 Alors l'affichage sera :
  - ☐ `i`
  - ☐ `ABCDEF`
  - ☐ `cccccc`
  - ☐ `A`

6. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ `**** * * * * *`
- ☐ `** * * * * *`
- ☐ `***** * * * *`
- ☐ `** * * * * *`

7. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou

8. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque

9. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`

10. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ `0 1 2 3 0 1 2`
- ☐ `0 0 1 1 2 2 3`
- ☐ `0 1 2 0 1 2 3`
- ☐ `0 1 2 0 1 2`

11. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur Majeur

12. Si  $x$  est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1
- ☐ 1.5
- ☐ 0
- ☐ 0.5

13. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(struct date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `int afficher_date(date_s d);`

14. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Mineur Majeur
- ☐ rien
- ☐ Majeur

15. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses champs
- ☐ ses chants
- ☐ ses blocs

16. Le langage C est un langage

- ☐ composé
- ☐ interprété
- ☐ compilé
- ☐ lu, écrit, parlé

17. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 4
- ☐ 0
- ☐ 16

18. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 0
- ☐ 4
- ☐ 8

19. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 5
- ☐ 101
- ☐ 4

20. On considère deux variables booléennes A et B initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `A && B`
- ☐ `(!A || B)`
- ☐ `!(!A || B) == (A && !B)`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

2. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 4
- ☐ 3

3. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`

4. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C
- ☐ A
- ☐ B
- ☐ b

5. Un fichier source est :

- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document de référence du système
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document qui doit être protégé
- ☐ un document illisible pour les humains

6. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases
- ☐ ses chants
- ☐ ses champs
- ☐ ses blocs

7. Quel est l'opérateur de différence en C :

- ☐ `!=`
- ☐ `!`
- ☐ `<>`
- ☐ `≠`

8. Le type des réels en C est :

- ☐ `double`
- ☐ `real`
- ☐ `char`
- ☐ `int`

9. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée
- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir définie
- ☐ l'avoir déclarée et définie

10. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il ne compile pas

11. Un registre du processeur est :

- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

12. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `char afficher_menu(printf("menu"));`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `double afficher_menu();`

13. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 101
- ☐ 5
- ☐ 4

14. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=8, a=3, b=5

15. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 16
- ☐ 3
- ☐ 6
- ☐ 20

16. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
```

```
for(j=i;j<6;j=j+1)
{
    printf("*");
}
printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\* \*\*
- ☐ \*\* \*\* \*\* \*\*
- ☐ \*\* \*\*\* \*\*
- ☐ \*\*\*\* \*\*

17. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction main(), alors on peut écrire en début de main() :

- ☐ struct toto\_s toto;
- ☐ int toto.n = 3;
- ☐ toto\_s struct z = {3, 0.5};
- ☐ toto\_s n, x;
- ☐ int struct toto\_s = {3, -1e10};

18. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur **#include** manquante

19. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ des processus
- ☐ certaines données de la mémoire de travail
- ☐ les fichiers du disque
- ☐ en temps d'accès

20. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Mineur
- ☐ Mineur Majeur
- ☐ Majeur

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 4  
☐ j = 0  
☐ j = 5  
☐ j = %d

2. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants  
☐ ses champs  
☐ ses blocs  
☐ ses cases

3. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main  
☐ un ordre quelconque  
☐ alphabétique  
☐ dans lequel vous avez déclaré ces fonction

4. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 4  
☐ 3  
☐ 0

5. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`  
☐ `#include <studio.h>`  
☐ `#appart <stdlib.h>`  
☐ `#include <studlib.h>`

6. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`  
☐ `int factorielle(double n);`  
☐ `int factorielle(int x);`  
☐ `int factorielle();`

7. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction appelée avant sa déclaration  
☐ une directive préprocesseur `#include` manquante

8. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien  
☐ comportent une erreur qui ne sera pas détectée  
☐ ne comportent aucune erreur  
☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique

9. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(3,5)=8, a=3, b=5  
☐ f(a,b)=13, a=8, b=5  
☐ f(a,b)=8, a=8, b=5  
☐ f(a,b)=8, a=3, b=5

10. Vous utilisez une boucle **while** quand :

- ☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous n'avez pas déclaré de fonction  
☐ vous avez déjà fait un **for** dans le même programme principal  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

11. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4
- ☐ 0 1 2 3
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0

12. Sous unix (ou linux), la commande `ls` permet de :

- ☐ afficher le contenu d'un fichier texte
- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ voir des clips musicaux
- ☐ compiler un programme

13. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ j = 5
- ☐ j = %d
- ☐ j = 4
- ☐ j = 0

14. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur

15. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(int p, int q);`
- ☐ `int pgcd(2);`
- ☐ `int n = pgcd();`
- ☐ `n = pgcd(n, 3);`

16. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un fichier texte
- ☐ ouvrir un fichier texte
- ☐ créer un répertoire
- ☐ changer de répertoire courant

17. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%x et y=%y\n");`

18. Le type des réels en C est :

- ☐ `double`
- ☐ `real`
- ☐ `char`
- ☐ `int`

19. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y = 3;
13
14     x = y;
15
16     ...
17 }
```

- ☐ la variable `y` vaut 5
- ☐ la variable `x` vaut 5 et la variable `y` vaut 3
- ☐ la variable `x` vaut 3
- ☐ le programme affiche "Faux"

20. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`
- ☐ `a<=n<=b`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<n) || (n>b)`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int tab[] = 5;`  
☐ `int toto[taille=5];`  
☐ `int[] new tableau(5);`  
☐ `int toto[5];`  
☐ `char tableau[5];`

2. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1 0 1  
☐ 1 2 1 2 3  
☐ 0 0 0 1 1 1  
☐ 0 1 2 0 1 2

3. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants  
☐ ses blocs  
☐ ses cases  
☐ ses champs

4. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
```

```
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 5  
☐ j = 0  
☐ j = %d  
☐ j = 4

5. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`  
☐ `int exposant(double n, int x);`  
☐ `exposant(double x, int n, int r);`  
☐ `double exposant(double x, int n);`

6. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 5  
☐ 4  
☐ 1  
☐ 0

7. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b  
☐ C  
☐ A  
☐ B

8. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique  
☐ dans lequel ces fonctions sont appelées dans le main  
☐ un ordre quelconque  
☐ dans lequel vous avez déclaré ces fonction

9. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu();`  
☐ `int afficher_menu(int char);`  
☐ `char afficher_menu(printf("menu"));`  
☐ `void afficher_menu();`  
☐ `double afficher_menu();`

10. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int x, y);`  
☐ `int pgcd(int x, int x);`  
☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int y, int x);`

11. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée et définie  
☐ l'avoir définie  
☐ l'avoir déclarée  
☐ avoir défini une constante symbolique de la taille de cette fonction

12. Le type des réels en C est :

- ☐ `char`  
☐ `int`  
☐ `double`  
☐ `real`

13. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Mineur  
Majeur
- ☐ Majeur
- ☐ rien

14. Un fichier source est :

- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document illisible pour les humains
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document qui doit être protégé
- ☐ un document de référence du système

15. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc -Wall prog.exe -o prog.c`

16. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#appart <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <stdio.h>`
- ☐ `#include <studlib.h>`

17. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse syntaxique
- ☐ analyse lexicale
- ☐ analyse sémantique

18. Si cet avertissement apparaît à la compilation :

- warning: implicit declaration of function 'max'**  
, que doit-on chercher dans le programme ?
- ☐ une directive préprocesseur `#include` manquante
  - ☐ un désaccord entre la déclaration et la définition d'une fonction

- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration

19. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 0
- ☐ 42
- ☐ 6
- ☐ 1

20. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5

2. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ loop i;
- ☐ int %d;
- ☐ int loop n;
- ☐ int k;

3. Au début de la fonction main() on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ A
- ☐ cccccc
- ☐ i

4. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

5. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 6
- ☐ 1
- ☐ 0
- ☐ 42

6. Pour déclarer une fonction pgcd qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ void pgcd(int x, int y);
- ☐ int pgcd(int x, y);
- ☐ int pgcd(int x, int x);
- ☐ int pgcd(int y, int x);

7. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 3
- ☐ 4
- ☐ 5

8. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1
- ☐ 1 2 3 1 2
- ☐ 0 1 2 0 1 2
- ☐ 0 0 1 1 2 2

9. Pour compiler un programme prog.c, on utilise la ligne de commande :

- ☐ gcc prog.c -o -Wall prog.exe
- ☐ gcc -Wall prog.exe -o prog.c
- ☐ gcc -Wall prog.c -o prog.exe
- ☐ gcc prog.exe -Wall -o prog.c

10. Vous utilisez une boucle while quand :

- ☐ vous avez déjà fait un for dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction

11. Si x est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1.5
- ☐ 0
- ☐ 0.5
- ☐ 1

12. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `toto_s n, x;`
- ☐ `int toto.n = 3;`
- ☐ `struct toto_s toto;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`

13. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C
- ☐ B
- ☐ b
- ☐ A

14. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ tour à tour, un petit peu à chaque fois
- ☐ tous ensemble
- ☐ en parallèle, chacun dans un registre

15. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 16
- ☐ le terminal affiche 8
- ☐ le bus explose
- ☐ la case mémoire 8 contiendra 0

16. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal

17. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée
- ☐ l'avoir déclarée et définie

- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir définie

18. Un fichier source est :

- ☐ un document de référence du système
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un document qui doit être protégé
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document illisible pour les humains

19. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses blocs
- ☐ ses cases
- ☐ ses champs

20. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ répéter un bloc tant qu'une condition est vérifiée
- ☐ retourner un bloc
- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ mettre les blocs en séquence les uns à la suite des autres

**Barème : 1 points par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction

2. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`

3. Un bit est :

- ☐ un battement d'horloge processeur
- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)

4. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a = b`
- ☐ `a == b`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a{n, x} == b{n, x}`

5. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(n);`
- ☐ `int n = carre();`
- ☐ `n = carre(int n);`
- ☐ `int carre(2);`

6. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ certaines données de la mémoire de travail
- ☐ des processus
- ☐ les fichiers du disque
- ☐ en temps d'accès

7. Si cette erreur apparaît à la compilation :  
`error: expected ';' before '}' token` que doit-on chercher dans le programme ?

- ☐ une accolade manquante
- ☐ un point-virgule manquant
- ☐ une accolade en trop
- ☐ un point-virgule en trop

8. Quel est l'opérateur de différence en C :

- ☐ `≠`
- ☐ `!=`
- ☐ `!`
- ☐ `<>`

9. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur  
Majeur

10. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ `b`
- ☐ `B`
- ☐ `C`
- ☐ `A`

11. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle `[a..b]`, on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<n) || (n>b)`
- ☐ `a<=n<=b`

12. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42
- ☐ 0
- ☐ 1
- ☐ 6

13. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :
- ☐ `x = mccarthy(n);`
  - ☐ `int mccarthy(int 2);`
  - ☐ `n = mccarthy(p, q);`
  - ☐ `n = mccarthy();`
14. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.
- ☐ `#define taille = 3`
  - ☐ `#define N 3`
  - ☐ `#define N = 3`
  - ☐ `#define taille = N`
15. Le langage C est un langage
- ☐ composé
  - ☐ compilé
  - ☐ lu, écrit, parlé
  - ☐ interprété
16. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :
- ☐ `void saisie_utilisateur(int n);`
  - ☐ `saisie_utilisateur(scanf(%d));`
  - ☐ `int saisie_utilisateur();`
  - ☐ `void saisie_utilisateur(char c);`

17. Quel est le problème d'un programme comportant les lignes suivantes ?
- ```
while (1)
{
    printf("coucou\n");
}
```
- ☐ il n'affiche rien
  - ☐ il comporte une boucle infinie
  - ☐ il ne compile pas
  - ☐ il risque d'afficher bonjour à la place de coucou
18. Pour l'extrait de programme suivant :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```
- qu'est ce qui sera affiché ?
- ☐ 0 1 2 0 1 2 3
  - ☐ 0 0 1 1 2 2 3
  - ☐ 0 1 2 3 0 1 2
  - ☐ 0 1 2 0 1 2

19. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
- ☐ `int pgcd(int y, int x);`
  - ☐ `int pgcd(int x, int x);`
  - ☐ `int pgcd(int x, y);`
  - ☐ `void pgcd(int x, int y);`
20. Soit la fonction `g` définie par :
- ```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```
- Alors l'expression `g(0)` prendra la valeur :
- ☐ 7
  - ☐ 0
  - ☐ 5

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17 }
```

- ☐ la variable y vaut 5
- ☐ la variable x vaut 5 et la variable y vaut 3
- ☐ la variable x vaut 3
- ☐ le programme affiche "Faux"

2. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 8
- ☐ 7
- ☐ 111

3. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a = b
- ☐ a{n, x} == b{n, x}
- ☐ a == b
- ☐ (a.n == b.n) && (a.x == b.x)

4. L'ordonnancement par tourniquet permet :

- ☐ d'afficher des ronds colorés à l'écran
- ☐ de doubler la mémoire disponible
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'entretenir l'illusion que les processus tournent en parallèle

5. Quels calculs peut-on programmer en programmation structurée ?

- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée

6. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Mineur Majeur
- ☐ Mineur
- ☐ Majeur

7. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`
- ☐ `int n = carre();`
- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`

8. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5

9. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ ouvrir un fichier texte
- ☐ créer un répertoire
- ☐ changer de répertoire courant
- ☐ créer un fichier texte

10. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<n) || (n>b)`

11. Vous utilisez une boucle `while` quand :
- ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ vous n'avez pas déclaré de fonction
12. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :
- ☐ 0.5
  - ☐ 1
  - ☐ 0
  - ☐ 1.5
13. Pour compiler un programme `prog.c`, on utilise la ligne de commande :
- ☐ `gcc -Wall prog.exe -o prog.c`
  - ☐ `gcc prog.exe -Wall -o prog.c`
  - ☐ `gcc -Wall prog.c -o prog.exe`
  - ☐ `gcc prog.c -o -Wall prog.exe`
14. Quel est le problème d'un programme comportant les lignes suivantes ?
- ```
while (1)
{
    printf("coucou\n");
}
```
- ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il ne compile pas
  - ☐ il n'affiche rien
  - ☐ il comporte une boucle infinie

15. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
`Undefined symbols : "_printf" ou référence indéfinie vers < printf >`
- ☐ l'analyse harmonique
  - ☐ l'analyse des entrées clavier
  - ☐ l'analyse sémantique
  - ☐ l'édition de liens
16. Soit la fonction `f` définie par :
- ```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```
- Alors l'expression `f(0)` prendra la valeur :
- ☐ 0
  - ☐ 3
  - ☐ 4
17. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :
- ☐ `saisie_utilisateur(scanf(%d));`
  - ☐ `int saisie_utilisateur();`
  - ☐ `void saisie_utilisateur(int n);`
  - ☐ `void saisie_utilisateur(char c);`

18. Sous unix (ou linux), la commande `cd` permet de :
- ☐ ouvrir un bureau partagé (common desktop)
  - ☐ récupérer un programme arrêté avec la commande `ab`
  - ☐ détruire un fichier
  - ☐ jouer de la musique
  - ☐ changer de répertoire courant
19. Les lignes
- ```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```
- ☐ comportent une erreur qui ne sera pas détectée
  - ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
  - ☐ ne comportent aucune erreur
  - ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
20. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
- ☐ `for(i=1;i<5;i=i+1)`
  - ☐ `for(i=1;i<=5;i=i+1)`
  - ☐ `for(i=0;i<5;i=i+1)`
  - ☐ `for(i=0;i<=5;i=i+1)`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution du programme :

```
1 lecture 8 r0
2 valeur 3 r1
3 mult r1 r0
4 valeur 1 r2
5 add r2 r0
6 ecriture r0 8
7 stop
8 5
```

- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 16
- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose

2. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5

3. Pour compiler un programme prog.c, on utilise la ligne de commande :

- ☐ gcc -Wall prog.exe -o prog.c
- ☐ gcc -Wall prog.c -o prog.exe
- ☐ gcc prog.exe -Wall -o prog.c
- ☐ gcc prog.c -o -Wall prog.exe

4. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a == b
- ☐ a{n, x} == b{n, x}
- ☐ a = b
- ☐ (a.n == b.n) && (a.x == b.x)

5. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 4
- ☐ 101
- ☐ 5
- ☐ 3

6. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.

- ☐ #define N = 3
- ☐ #define taille = 3
- ☐ #define taille = N
- ☐ #define N 3

7. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2 0
- ☐ 8 2

8. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses cases
- ☐ ses champs
- ☐ ses blocs

9. Si factorielle est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ n = factorielle();
- ☐ n = factorielle(p, q);
- ☐ printf("%d", factorielle(n));
- ☐ int factorielle(int 2);

10. Pour déclarer une fonction factorielle qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ struct int factorielle(int n);
- ☐ int factorielle();
- ☐ int factorielle(double n);
- ☐ int factorielle(int x);

11. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 20
- ☐ 3
- ☐ 16

12. Si x est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1.5
- ☐ 0
- ☐ 0.5
- ☐ 1

13. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 4
- ☐ 8
- ☐ 0
- ☐ 16

14. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ i
- ☐ cccccc
- ☐ A

15. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas

16. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale
- ☐ analyse syntaxique
- ☐ analyse sémantique
- ☐ analyse harmonique

17. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ en parallèle, chacun dans un registre
- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ tour à tour, un petit peu à chaque fois
- ☐ tous ensemble

18. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <stdlib.h>`
- ☐ `#appart <stdlib.h>`

19. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

20. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(2/3);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(x * x) - racine(x);`



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Un fichier source est :
  - ☐ un document de référence du système
  - ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
  - ☐ un document illisible pour les humains
  - ☐ un document qui doit être protégé
  - ☐ un fichier texte qui sera traduit en instructions processeur
- Un programme en langage C doit comporter une et une seule définition de la fonction :
  - ☐ begin
  - ☐ include
  - ☐ init
  - ☐ main
- Dans la commande gcc, l'option `-Wall` signifie :
  - ☐ qu'il faut lancer un débogueur
  - ☐ que l'on veut voir tous les avertissements
  - ☐ qu'il faut indenter le fichier source
  - ☐ qu'on veut changer aléatoirement de fond d'écran
- Un enregistrement permet de grouper plusieurs valeurs dans :
  - ☐ ses blocs
  - ☐ ses champs
  - ☐ ses cases
  - ☐ ses chants
- Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :
  - ☐ `int factorielle(int x);`
  - ☐ `int factorielle(double n);`
  - ☐ `int factorielle();`
  - ☐ `struct int factorielle(int n);`

- Si  $x$  est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :
  - ☐ 0
  - ☐ 0.5
  - ☐ 1
  - ☐ 1.5
- Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
  - ☐ `for(i=1; i<5; i=i+1)`
  - ☐ `for(i=0; i<=5; i=i+1)`
  - ☐ `for(i=0; i<5; i=i+1)`
  - ☐ `for(i=1; i<=5; i=i+1)`
- L'écriture 111 en binaire correspond au nombre naturel :
  - ☐ 3
  - ☐ 8
  - ☐ 7
  - ☐ 111
- Soit la fonction  $f$  définie par :
 

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

 Alors l'expression  $f(1)$  prendra la valeur :
  - ☐ 5
  - ☐ 0
  - ☐ 1
  - ☐ 4

- Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4; i>0; i=i-1)
{
    for(j=i; j<6; j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\*\* \*\*\*\* \*\*\* \*\*
- ☐ \*\*\*\* \*\*\*\*\* \*\*\*\*
- ☐ \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*
- ☐ \*\* \*\*\* \*\*\*\*\*

- Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :
  - ☐ `double afficher_menu();`
  - ☐ `char afficher_menu(printf("menu"));`
  - ☐ `int afficher_menu();`
  - ☐ `void afficher_menu();`
  - ☐ `int afficher_menu(int char);`
- Sous unix (ou linux), la commande `ls` permet de :
  - ☐ afficher le contenu d'un fichier texte
  - ☐ voir des clips musicaux
  - ☐ compiler un programme
  - ☐ afficher la liste de fichiers contenus dans un répertoire
- Une *segmentation fault* est une erreur qui survient lorsque :
  - ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
  - ☐ la division du programme en zones homogènes échoue

- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée

14. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `char c;`
- ☐ `int char;`
- ☐ `char 'c';`
- ☐ `char "c";`

15. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4

16. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme?

- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction

17. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`
- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`
- ☐ `int n = carre();`

18. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 16
- ☐ 20

- ☐ 6
- ☐ 3

19. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int pgcd(2);`
- ☐ `n = pgcd(n, 3);`
- ☐ `int n = pgcd();`
- ☐ `n = pgcd(int p, int q);`

20. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur  
Majeur

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :
  - ☐ certaines données de la mémoire de travail
  - ☐ des processus
  - ☐ les fichiers du disque
  - ☐ en temps d'accès
- Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :
  - ☐ sélectionner entre deux blocs à l'aide d'une condition
  - ☐ répéter un bloc tant qu'une condition est vérifiée
  - ☐ mettre les blocs en séquence les uns à la suite des autres
  - ☐ retourner un bloc
- Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :
  - ☐ Mineur
  - ☐ Majeur
  - ☐ Majeur
  - ☐ Mineur
  - ☐ rien
- Le type des réels en C est :
  - ☐ int
  - ☐ real
  - ☐ char
  - ☐ double

- Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
  - ☐ dans lequel ces fonctions sont appelées dans le main
  - ☐ un ordre quelconque
  - ☐ alphabétique
  - ☐ dans lequel vous avez déclaré ces fonction
- Un programme en langage C doit comporter une et une seule définition de la fonction :
  - ☐ include
  - ☐ init
  - ☐ begin
  - ☐ main
- Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :
  - ☐ 0
  - ☐ 3
  - ☐ 4
- Avant de faire appel à une fonction il est nécessaire de :
  - ☐ avoir défini une constante symbolique de la taille de cette fonction
  - ☐ l'avoir déclarée et définie
  - ☐ l'avoir définie
  - ☐ l'avoir déclarée

9. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 16
- ☐ 6
- ☐ 3
- ☐ 20

- Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?
  - ☐ une fonction appelée avant sa déclaration
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction déclarée mais non définie
- Un fichier source est :
  - ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
  - ☐ un document qui doit être protégé
  - ☐ un fichier texte qui sera traduit en instructions processeur
  - ☐ un document illisible pour les humains
  - ☐ un document de référence du système
- Vous utilisez une boucle `while` quand :
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ l'incrément de la variable de boucle n'est pas 1
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous n'avez pas déclaré de fonction

13. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
- ☐ `printf("Valeur de n ? %g\n", n);`
  - ☐ un débogueur
  - ☐ `printf("Valeur de n ? %d\n", n);`
  - ☐ `scanf("%d", &n);`
14. Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`, que doit-on chercher dans le programme ?
- ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction déclarée mais non définie
  - ☐ une fonction appelée avant sa déclaration
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
15. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?
- ☐ `(!A || B)`
  - ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `!(A || B) == (A && !B)`
  - ☐ `A && B`

16. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :
- ☐ `int factorielle(double n);`
  - ☐ `struct int factorielle(int n);`
  - ☐ `int factorielle(int x);`
  - ☐ `int factorielle();`
17. Quel est l'opérateur de différence en C :
- ☐ `≠`
  - ☐ `!=`
  - ☐ `!`
  - ☐ `<>`
18. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :
- ☐ analyse syntaxique
  - ☐ analyse harmonique
  - ☐ analyse sémantique
  - ☐ analyse lexicale
19. Soit le programme principal suivant :
- ```
int main()
{
    int a = 3;
```

```
int b = 5;
printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

20. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses chants
  - ☐ ses blocs
  - ☐ ses champs
  - ☐ ses cases

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tour à tour, un petit peu à chaque fois
- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ tous ensemble
- ☐ en parallèle, chacun dans un registre

2. Un fichier source est :

- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document qui doit être protégé
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
- ☐ un document illisible pour les humains
- ☐ un document de référence du système

3. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`
- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`

4. L'ordonnancement par tourniquet permet :

- ☐ de doubler la mémoire disponible
- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ d'afficher des ronds colorés à l'écran
- ☐ de ne pas perdre de temps avec la commutation de contexte

5. Soit la fonction  $f$  définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression  $f(1)$  prendra la valeur :

- ☐ 4
- ☐ 5
- ☐ 1
- ☐ 0

6. Soit la fonction  $f$  définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression  $f(0)$  prendra la valeur :

- ☐ 0
- ☐ 3
- ☐ 4

7. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur
- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `scanf("%d", &n);`

8. Si  $a$  et  $b$  sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de  $a$  et de  $b$  on utilise la condition :

- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a == b`

9. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction  $f$  ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

10. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que  $x$  est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(x * x) - racine(x);`

11. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?
- ☐ une fonction appelée avant sa déclaration
  - ☐ une directive préprocesseur **#include** manquante
  - ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
12. Le code suivant :
- ```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```
- affichera :
- ☐ Mineur
  - ☐ Mineur Majeur
  - ☐ rien
  - ☐ Majeur
13. Le bus système sert à :
- ☐ Écrire des données sur le disque dur
  - ☐ Arriver à l'heure en cours
  - ☐ transporter les processus du tourniquet au processeur
  - ☐ Transférer des données et intructions entre processeur et mémoire

14. Quels calculs peut-on programmer en programmation structurée ?
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
  - ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
  - ☐ certains programmes sont de vrais plats de spaghetti
  - ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
15. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses champs
  - ☐ ses cases
  - ☐ ses blocs
  - ☐ ses chants
16. Pour l'extrait de programme suivant :
- ```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```
- La valeur affichée est :
- ☐ 16
  - ☐ 0
  - ☐ 8
  - ☐ 4

17. Si **factorielle** est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
- ☐ `printf("%d", factorielle(n));`
  - ☐ `int factorielle(int 2);`
  - ☐ `n = factorielle(p, q);`
  - ☐ `n = factorielle();`
18. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
- ☐ `int char;`
  - ☐ `char c;`
  - ☐ `char "c";`
  - ☐ `char 'c';`
19. Pour l'extrait de programme suivant :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```
- qu'est ce qui sera affiché ?
- ☐ 0 1 2 0 1 2
  - ☐ 0 1 2 0 1 2 3
  - ☐ 0 0 1 1 2 2 3
  - ☐ 0 1 2 3 0 1 2
20. Si **x** est une variable réelle (de type double) alors **x = 3/2** lui affecte la valeur :
- ☐ 1.5
  - ☐ 0
  - ☐ 1
  - ☐ 0.5

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5  
☐ 0  
☐ 7

2. Le bus système sert à :

- ☐ Écrire des données sur le disque dur  
☐ Arriver à l'heure en cours  
☐ Transférer des données et instructions entre processeur et mémoire  
☐ transporter les processus du tourniquet au processeur

3. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante  
☐ une fonction appelée avant sa déclaration  
☐ une fonction déclarée mais non définie  
☐ un désaccord entre la déclaration et la définition d'une fonction

4. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1; i<5; i=i+1)`  
☐ `for(i=0; i<5; i=i+1)`  
☐ `for(i=0; i<=5; i=i+1)`  
☐ `for(i=1; i<=5; i=i+1)`

5. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs  
☐ ses chants  
☐ ses cases  
☐ ses blocs

6. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`  
☐ `int n = pgcd();`  
☐ `int pgcd(2);`  
☐ `n = pgcd(int p, int q);`

7. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie  
☐ il risque d'afficher bonjour à la place de coucou  
☐ il n'affiche rien  
☐ il ne compile pas

8. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`  
☐ `(a<n) || (n>b)`  
☐ `(n<=a) && (n<=b)`  
☐ `a<=n<=b`

9. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4  
☐ 0 1 2 3  
☐ 4 3 2 1  
☐ 4 3 2 1 0

10. Quels calculs peut-on programmer en programmation structurée ?

- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine  
☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine  
☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée  
☐ certains programmes sont de vrais plats de spaghetti

11. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(A == TRUE) && (B == TRUE)`  
☐ `A && B`  
☐ `(!A || B)`  
☐ `!(A || B) == (A && !B)`

12. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 1 2 3 4 5 6
- ☐ 0 2 4 6

13. Sous unix (ou linux), la commande `ls` permet de :

- ☐ afficher le contenu d'un fichier texte
- ☐ voir des clips musicaux
- ☐ afficher la liste de fichiers contenus dans un répertoire
- ☐ compiler un programme

14. Le type des réels en C est :

- ☐ `int`
- ☐ `real`
- ☐ `double`
- ☐ `char`

15. Un registre du processeur est :

- ☐ un composant qui contient la liste des fichiers du système
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

16. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.c -o prog.exe`

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

18. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
```

```
        printf("%d ", j);
```

```
    }
```

```
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 3 0 1 2
- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2

19. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 1
- ☐ 0
- ☐ 0.5
- ☐ 1.5

20. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique
- ☐ dans lequel ces fonctions sont appelées dans le `main`



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un bit est :

- ☐ l'instruction qui met fin à un programme
- ☐ un battement d'horloge processeur
- ☐ la longueur d'un mot mémoire
- ☐ un chiffre binaire (0 ou 1)

2. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = 0
- ☐ j = %d
- ☐ j = 5
- ☐ j = 4

3. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max' ,** que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie

4. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
```

```
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

5. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

6. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#include <stdlib.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studio.h>`

7. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il comporte une boucle infinie

8. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ transporter les processus du tourniquet au processeur
- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur

9. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 3 0 1 2
- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 0 1 2

10. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `int n = pgcd();`
- ☐ `int pgcd(2);`

11. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `char "c";`
- ☐ `int char;`

12. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?
- ☐ un point-virgule en trop
  - ☐ un point-virgule manquant
  - ☐ une accolade en trop
  - ☐ une accolade manquante
13. L'écriture 111 en binaire correspond au nombre naturel :
- ☐ 7
  - ☐ 111
  - ☐ 3
  - ☐ 8
14. Pour déclarer une fonction **saisie\_utilisateur** qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :
- ☐ `void saisie_utilisateur(char c);`
  - ☐ `void saisie_utilisateur(int n);`
  - ☐ `int saisie_utilisateur();`
  - ☐ `saisie_utilisateur(scanf(%d));`
15. Sur unix (ou linux), la commande **mkdir** permet de :
- ☐ créer un fichier texte
  - ☐ ouvrir un fichier texte
  - ☐ changer de répertoire courant
  - ☐ créer un répertoire
16. Soit le programme principal suivant :
- ```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction **f** ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`

17. Lorsqu'un programme utilise **printf** ou **scanf** il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#include <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#appart <stdlib.h>`

18. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ `j = 0`
- ☐ `j = %d`
- ☐ `j = 5`
- ☐ `j = 4`

19. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée
- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir définie
- ☐ l'avoir déclarée et définie

20. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ `***** ***** *** **`
- ☐ `** *** *****`
- ☐ `**** *****`
- ☐ `** ** ** ** **`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou

2. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=8, b=5

3. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a{n, x} == b{n, x}
- ☐ a == b
- ☐ a = b
- ☐ (a.n == b.n) && (a.x == b.x)

4. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

- ☐ printf("x=%d et y=%d\n",x,y);
- ☐ printf("x=%d et y=%d\n",x y);
- ☐ printf("x=%x et y=%y\n");
- ☐ printf("x=%d et y=%d\n,x,y");

5. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
```

printf("\n");

affichera :

- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0

6. On souhaite faire une boucle de contrôle de saisie : tant que l'entier n n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de n. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition cond :

- ☐ a<=n<=b
- ☐ (a<n) || (n>b)
- ☐ (a<=n) && (n<=b)
- ☐ (n<=a) && (n<=b)

7. Si factorielle est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ n = factorielle(p, q);
- ☐ printf("%d", factorielle(n));
- ☐ n = factorielle();
- ☐ int factorielle(int 2);

8. Une variable booléenne est un variable :

- ☐ réelle positive
- ☐ qui est vraie ou fausse
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ jamais nulle
- ☐ à laquelle une valeur vient d'être affectée

9. Dans la commande gcc, l'option -Wall signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut lancer un débogueur
- ☐ qu'il faut indenter le fichier source

10. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses cases
- ☐ ses champs
- ☐ ses blocs

11. Le type des réels en C est :

- ☐ char
- ☐ int
- ☐ real
- ☐ double

12. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

- ☐ la variable x vaut 5 et la variable y vaut 3
- ☐ la variable x vaut 3
- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"

13. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ tour à tour, un petit peu à chaque fois
- ☐ en parallèle, chacun dans un registre
- ☐ tous ensemble

14. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche "Faux"
- ☐ le terminal affiche x = 5
- ☐ le terminal affiche 5
- ☐ le terminal affiche x = 2

15. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée
- ☐ l'avoir déclarée et définie
- ☐ l'avoir définie
- ☐ avoir défini une constante symbolique de la taille de cette fonction

16. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours
- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur

17. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 4
- ☐ j = 0
- ☐ j = 5
- ☐ j = %d

18. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ main
- ☐ init
- ☐ begin
- ☐ include

19. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(n, 3);`
- ☐ `int pgcd(2);`
- ☐ `int n = pgcd();`
- ☐ `n = pgcd(int p, int q);`

20. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3
- ☐ 20
- ☐ 6
- ☐ 16

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(int p, int q);`  
☐ `int pgcd(2);`  
☐ `int n = pgcd();`  
☐ `n = pgcd(n, 3);`

2. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses cases  
☐ ses blocs  
☐ ses champs  
☐ ses chants

3. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`  
☐ `char "c";`  
☐ `char c;`  
☐ `char 'c';`

4. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule en trop  
☐ un point-virgule manquant  
☐ une accolade manquante  
☐ une accolade en trop

5. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 16  
☐ le bus explose  
☐ le terminal affiche 8  
☐ la case mémoire 8 contiendra 0

6. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »**

- ☐ l'analyse sémantique  
☐ l'analyse des entrées clavier  
☐ l'analyse harmonique  
☐ l'édition de liens

7. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `init`  
☐ `main`  
☐ `include`  
☐ `begin`

8. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(3,5)=8, a=3, b=5`  
☐ `f(a,b)=13, a=8, b=5`  
☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=8, a=3, b=5`

9. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique

- ☐ un ordre quelconque  
☐ dans lequel ces fonctions sont appelées dans le `main`  
☐ dans lequel vous avez déclaré ces fonction

10. Si cet avertissement apparaît à la compilation : **warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration  
☐ une directive préprocesseur `#include` manquante  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction déclarée mais non définie

11. Après exécution jusqu'à la ligne 15 du programme C :

```
10  ...
11  int main() {
12      int x = 5;
13
14      x = 3 * x + 1;
15
16      ...
17  }
```

- ☐ le programme affiche `x`  
☐ la variable `x` vaut  $-\frac{1}{2}$   
☐ la variable `x` vaut 16  
☐ le programme affiche **\*\*\*\***

12. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42  
☐ 0  
☐ 6  
☐ 1

13. Le langage C est un langage

- ☐ composé
- ☐ interprété
- ☐ lu, écrit, parlé
- ☐ compilé

14. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `void afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `char afficher_menu(sprintf("menu"));`
- ☐ `int afficher_menu();`
- ☐ `double afficher_menu();`

15. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`
- ☐ `int factorielle();`
- ☐ `int factorielle(double n);`
- ☐ `struct int factorielle(int n);`

16. Au début de la fonction `main()` on place le code :

```
char b = 'A';  
b = b + 2;  
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ C
- ☐ A
- ☐ B
- ☐ b

17. Pour l'extrait de programme suivant :

```
int i;  
int j;  
for(i=4;i>0;i=i-1)  
{  
    for(j=i;j<6;j=j+1)  
    {  
        printf("*");  
    }  
    printf(" ");  
}
```

qu'est ce qui sera affiché ?

- ☐ \*\*\*\* \* \* \* \*
- ☐ \*\* \* \* \* \*
- ☐ \*\*\*\*\* \* \* \*
- ☐ \*\* \* \* \* \*

18. Soit la fonction `g` définie par :

```
int g(int a)  
{  
    printf("a = \n", %d);  
    if (1 > 0)  
    {  
        return 5;  
    }  
    return 7;  
}
```

Alors l'expression `g(0)` prendra la valeur :

☐ 0

☐ 7

☐ 5

19. Si `a` et `b` sont deux variables de type :

```
struct toto_s  
{  
    int n;  
    double x;  
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a{n, x} == b{n, x}`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a = b`
- ☐ `a == b`

20. Sous unix (ou linux), la commande `cd` permet de :

- ☐ jouer de la musique
- ☐ ouvrir un bureau partagé (common desktop)
- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ changer de répertoire courant
- ☐ détruire un fichier

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche \*\*\*\*
- ☐ le programme affiche x
- ☐ la variable x vaut  $-\frac{1}{2}$
- ☐ la variable x vaut 16

2. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque
- ☐ alphabétique

3. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 0 1 2 3 4 5 6
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6

4. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <studlib.h>`

5. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`

6. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur
- ☐ que l'on veut voir tous les avertissements

7. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char c;`

8. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A
- ☐ C
- ☐ B
- ☐ b

9. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1

10. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction

11. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`
- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`
- ☐ `exposant(double x, int n, int r);`

12. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int pgcd(2);`
- ☐ `int n = pgcd();`
- ☐ `n = pgcd(int p, int q);`
- ☐ `n = pgcd(n, 3);`

13. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `toto_s n, x;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `struct toto_s toto;`
- ☐ `int toto.n = 3;`

14. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ `**** * **** * **** *`
- ☐ `** * ** * ** * ** *`
- ☐ `** * ** * **** * **** *`
- ☐ `***** * **** * ** * **** *`

15. Sous unix (ou linux), la commande `cd` permet de :

- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ ouvrir un bureau partagé (common desktop)
- ☐ détruire un fichier
- ☐ changer de répertoire courant
- ☐ jouer de la musique

16. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses champs
- ☐ ses chants
- ☐ ses cases

17. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `int saisie_utilisateur();`

18. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 6
- ☐ 0
- ☐ 42
- ☐ 1

19. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 0 1 2 3 4

20. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur
- ☐ Arriver à l'heure en cours

2. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 0 1 0 1
- ☐ 0 1 2 0 1 2
- ☐ 1 2 3 1 2
- ☐ 0 0 1 1 2 2

3. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<n) || (n>b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<=n) && (n<=b)`
- ☐ `a<=n<=b`

4. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`

5. Sous unix (ou linux), la commande `ls` permet de :

- ☐ compiler un programme
- ☐ voir des clips musicaux
- ☐ afficher le contenu d'un fichier texte
- ☐ afficher la liste de fichiers contenus dans un répertoire

6. Le langage C est un langage

- ☐ lu, écrit, parlé
- ☐ compilé
- ☐ interprété
- ☐ composé

7. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy(p, q);`
- ☐ `x = mccarthy(n);`
- ☐ `n = mccarthy();`
- ☐ `int mccarthy(int 2);`

8. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée
- ☐ l'avoir déclarée et définie
- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir définie

9. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il comporte une boucle infinie

10. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction

11. Sous unix (ou linux), la commande `cd` permet de :

- ☐ ouvrir un bureau partagé (common desktop)
- ☐ détruire un fichier
- ☐ récupérer un programme arrêté avec la commande `ab`
- ☐ jouer de la musique
- ☐ changer de répertoire courant

12. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 0
- ☐ 1
- ☐ 5
- ☐ 4

13. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`
- ☐ `int n = carre();`
- ☐ `int carre(2);`

14. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur

15. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
```

```
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Majeur

16. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 7

17. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`

18. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `struct toto_s toto;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `toto_s n, x;`
- ☐ `int toto.n = 3;`

19. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle(p, q);`
- ☐ `printf("%d", factorielle(n));`
- ☐ `int factorielle(int 2);`
- ☐ `n = factorielle();`

20. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char "c";`

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`  
☐ `saisie_utilisateur(scanf(%d));`  
☐ `int saisie_utilisateur();`  
☐ `void saisie_utilisateur(int n);`

2. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche 5  
☐ le terminal affiche `x = 2`  
☐ le terminal affiche `x = 5`  
☐ le terminal affiche "Faux"

3. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur  
☐ `printf("Valeur de n ? %g\n", n);`  
☐ `printf("Valeur de n ? %d\n", n);`  
☐ `scanf("%d", &n);`

4. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal  
☐ vous n'avez pas déclaré de fonction  
☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

5. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `init`  
☐ `include`  
☐ `main`  
☐ `begin`

6. Si cet avertissement apparaît à la compilation :  
**warning: implicit declaration of function 'max'**, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une fonction appelée avant sa déclaration  
☐ une fonction déclarée mais non définie

7. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(double n);`  
☐ `int factorielle(int x);`  
☐ `int factorielle();`  
☐ `struct int factorielle(int n);`

8. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 111  
☐ 3  
☐ 8  
☐ 7

9. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée  
☐ l'avoir définie  
☐ avoir défini une constante symbolique de la taille de cette fonction  
☐ l'avoir déclarée et définie

10. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou  
☐ il comporte une boucle infinie  
☐ il n'affiche rien  
☐ il ne compile pas

11. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque  
☐ alphabétique  
☐ dans lequel ces fonctions sont appelées dans le `main`  
☐ dans lequel vous avez déclaré ces fonction

12. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable `x` vaut 5 et la variable `y` vaut 0  
☐ la variable `x` vaut 0  
☐ le programme affiche "Faux"  
☐ la variable `y` vaut 5

13. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale  
☐ analyse sémantique  
☐ analyse harmonique  
☐ analyse syntaxique

14. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3
- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4

15. Si cette erreur apparaît à la compilation :

Undefined symbols : "\_printf" ou  
référence indéfinie vers « printf » que doit-on chercher dans le programme ?

- ☐ une faute de frappe dans un appel de fonction
- ☐ une directive préprocesseur #include manquante
- ☐ un caractère interdit en C
- ☐ une variable non déclarée

16. Un fichier source est :

- ☐ un document illisible pour les humains
- ☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur
- ☐ un document de référence du système

☐ un document qui doit être protégé

☐ un fichier texte qui sera traduit en instructions processeur

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Majeur
- ☐ Mineur
- ☐ Mineur Majeur

18. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

Undefined symbols : "\_printf" ou  
référence indéfinie vers « printf »

- ☐ l'analyse harmonique
- ☐ l'analyse sémantique

☐ l'analyse des entrées clavier

☐ l'édition de liens

19. Pour compiler un programme prog.c, on utilise la ligne de commande :

- ☐ gcc -Wall prog.c -o prog.exe
- ☐ gcc prog.c -o -Wall prog.exe
- ☐ gcc prog.exe -Wall -o prog.c
- ☐ gcc -Wall prog.exe -o prog.c

20. On souhaite faire une boucle de contrôle de saisie : tant que l'entier n n'appartient pas à l'intervalle [a..b], on recommence la saisie de n. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition cond :

- ☐ (a<n) || (n>b)
- ☐ (a<=n) && (n<=b)
- ☐ a<=n<=b
- ☐ (n<=a) && (n<=b)

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<5;i=i+1)`  
☐ `for(i=1;i<=5;i=i+1)`  
☐ `for(i=0;i<=5;i=i+1)`  
☐ `for(i=1;i<5;i=i+1)`

2. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements  
☐ qu'on veut changer aléatoirement de fond d'écran  
☐ qu'il faut indenter le fichier source  
☐ qu'il faut lancer un débogueur

3. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche `x = 5`  
☐ le terminal affiche `x = 2`  
☐ le terminal affiche "Faux"  
☐ le terminal affiche 5

4. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(struct date_s d);`  
☐ `void afficher_date(date_s d);`  
☐ `int afficher_date(date_s d);`  
☐ `struct date_s afficher_date(struct date_s d);`

5. Un registre du processeur est :

- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs  
☐ une gamme de fréquence de fonctionnement du processeur  
☐ une unité de calcul spécialisée de l'ordinateur  
☐ un composant qui contient la liste des fichiers du système

6. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2 3  
☐ 0 1 2 0 1 2 3  
☐ 0 1 2 0 1 2  
☐ 0 1 2 3 0 1 2

7. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `struct toto_s toto;`  
☐ `int toto.n = 3;`  
☐ `int struct toto_s = {3, -1e10};`  
☐ `toto_s n, x;`  
☐ `toto_s struct z = {3, 0.5};`

8. Vous utilisez une boucle `while` quand :

- ☐ l'incrément de la variable de boucle n'est pas 1  
☐ vous avez déjà fait un `for` dans le même programme principal  
☐ vous n'avez pas déclaré de fonction  
☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

9. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`  
☐ `x = racine(2/3);`  
☐ `x - 1 = racine(x);`  
☐ `x = racine(racine(x)*racine(x));`

10. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

Undefined symbols : "\_printf" ou référence indéfinie vers < printf >

- ☐ l'analyse des entrées clavier  
☐ l'édition de liens  
☐ l'analyse harmonique  
☐ l'analyse sémantique

11. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8  
☐ 3  
☐ 7  
☐ 111

12. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie  
☐ il risque d'afficher bonjour à la place de coucou  
☐ il ne compile pas  
☐ il n'affiche rien

13. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc prog.c -o -Wall prog.exe`

14. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 4
- ☐ 3

15. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
```

```
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 10
- ☐ 0
- ☐ 15
- ☐ 6

16. Une variable booléenne est un variable :

- ☐ réelle positive
- ☐ qui est vraie ou fausse
- ☐ jamais nulle
- ☐ à laquelle une valeur vient d'être affectée
- ☐ NaN (not a number, qui n'est pas un nombre)

17. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char c;`
- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `int char;`

18. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4

19. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `yppasswd`
- ☐ `mkdir TP4`
- ☐ `new TP4`
- ☐ `kwrite TP4`

20. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int y, int x);`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un bit est :
  - ☐ un chiffre binaire (0 ou 1)
  - ☐ un battement d'horloge processeur
  - ☐ l'instruction qui met fin à un programme
  - ☐ la longueur d'un mot mémoire
2. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :
  - ☐ `int mccarthy(int 2);`
  - ☐ `n = mccarthy(p, q);`
  - ☐ `n = mccarthy();`
  - ☐ `x = mccarthy(n);`
3. Pour compiler un programme `prog.c`, on utilise la ligne de commande :
  - ☐ `gcc -Wall prog.c -o prog.exe`
  - ☐ `gcc prog.exe -Wall -o prog.c`
  - ☐ `gcc -Wall prog.exe -o prog.c`
  - ☐ `gcc prog.c -o -Wall prog.exe`
4. Si cette erreur apparaît à la compilation :  
`Undefined symbols : "_printf" ou référence indéfinie vers « printf »` que doit-on chercher dans le programme ?
  - ☐ un caractère interdit en C
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une variable non déclarée
  - ☐ une faute de frappe dans un appel de fonction
5. Sous unix (ou linux), la commande `cd` permet de :
  - ☐ récupérer un programme arrêté avec la commande `ab`
  - ☐ détruire un fichier
  - ☐ changer de répertoire courant
  - ☐ jouer de la musique
  - ☐ ouvrir un bureau partagé (common desktop)

6. Sous unix (ou linux), la commande `ls` permet de :
  - ☐ voir des clips musicaux
  - ☐ compiler un programme
  - ☐ afficher le contenu d'un fichier texte
  - ☐ afficher la liste de fichiers contenus dans un répertoire
7. Un fichier source est :
  - ☐ un fichier texte qui sera traduit en instructions processeur
  - ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur
  - ☐ un document illisible pour les humains
  - ☐ un document qui doit être protégé
  - ☐ un document de référence du système
8. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :
  - ☐ 0
  - ☐ 3
  - ☐ 4
9. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
  - ☐ `void pgcd(int x, int y);`
  - ☐ `int pgcd(int x, int x);`
  - ☐ `int pgcd(int x, y);`
  - ☐ `int pgcd(int y, int x);`

10. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :
  - ☐ `(a.n == b.n) && (a.x == b.x)`
  - ☐ `a{n, x} == b{n, x}`
  - ☐ `a == b`
  - ☐ `a = b`
11. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :
  - ☐ `x = racine(2/3);`
  - ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x - 1 = racine(x);`
  - ☐ `x = racine(x * x) - racine(x);`
12. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction
  - ☐ `int toto[taille=5];`
  - ☐ `int toto[5];`
  - ☐ `char tableau[5];`
  - ☐ `int[] new tableau(5);`
  - ☐ `int tab[] = 5;`
13. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

  - ☐ il n'affiche rien
  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il ne compile pas
  - ☐ il comporte une boucle infinie

14. L'écriture 101 en binaire correspond au nombre naturel :
- ☐ 101
  - ☐ 5
  - ☐ 4
  - ☐ 3
15. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?
- ☐ `char c;`
  - ☐ `char "c";`
  - ☐ `int char;`
  - ☐ `char 'c';`
16. Sur unix (ou linux), la commande `mkdir` permet de :
- ☐ changer de répertoire courant
  - ☐ créer un répertoire
  - ☐ créer un fichier texte
  - ☐ ouvrir un fichier texte

17. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :
- ☐ `void saisie_utilisateur(char c);`
  - ☐ `int saisie_utilisateur();`
  - ☐ `saisie_utilisateur(scanf(%d));`
  - ☐ `void saisie_utilisateur(int n);`
18. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses champs
  - ☐ ses blocs
  - ☐ ses chants
  - ☐ ses cases
19. Le bus système sert à :
- ☐ transporter les processus du tourniquet au processeur
  - ☐ Écrire des données sur le disque dur

- ☐ Transférer des données et intructions entre processeur et mémoire
  - ☐ Arriver à l'heure en cours
20. Pour l'extrait de programme suivant :
- ```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```
- La valeur de somme affichée est :
- ☐ 6
  - ☐ 3
  - ☐ 16
  - ☐ 20



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir définie  
☐ avoir défini une constante symbolique de la taille de cette fonction  
☐ l'avoir déclarée  
☐ l'avoir déclarée et définie

2. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3  
☐ 0  
☐ 4

3. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 6 4 2 0  
☐ 8 2  
☐ 8 6 4 2  
☐ 0 2 4 6 8

4. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a = b`  
☐ `a{n, x} == b{n, x}`  
☐ `(a.n == b.n) && (a.x == b.x)`  
☐ `a == b`

5. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ que l'on veut voir tous les avertissements  
☐ qu'il faut indenter le fichier source  
☐ qu'on veut changer aléatoirement de fond d'écran  
☐ qu'il faut lancer un débogueur

6. L'ordonnancement par tourniquet permet :

- ☐ d'entretenir l'illusion que les processus tournent en parallèle  
☐ d'afficher des ronds colorés à l'écran  
☐ de ne pas perdre de temps avec la commutation de contexte  
☐ de doubler la mémoire disponible

7. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 0.5  
☐ 0  
☐ 1.5  
☐ 1

8. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
```

```
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2  
☐ 0 1 0 1 0 1 0 1  
☐ 0 0 0 1 1 1  
☐ 1 2 1 2 3

9. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`  
☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int x, y);`  
☐ `int pgcd(int x, int x);`

10. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs  
☐ ses cases  
☐ ses champs  
☐ ses chants

11. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(n<=a) && (n<=b)`  
☐ `(a<=n) && (n<=b)`  
☐ `a<=n<=b`  
☐ `(a<n) || (n>b)`

12. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
- ☐ `#include <stdlib.h>`
  - ☐ `#include <stdio.h>`
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <studio.h>`
13. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :
- ☐ `printf("x=%d et y=%d\n",x,y);`
  - ☐ `printf("x=%d et y=%d\n",x y);`
  - ☐ `printf("x=%x et y=%y\n");`
  - ☐ `printf("x=%d et y=%d\n",x,y);`
14. On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?
- ☐ `A && B`
  - ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `(!A || B)`
  - ☐ `!(A || B) == (A && !B)`
15. Vous utilisez une boucle `while` quand :
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ l'incrément de la variable de boucle n'est pas 1

16. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```
- ☐ la variable `y` vaut 5
  - ☐ le programme affiche "Faux"
  - ☐ la variable `x` vaut 0
  - ☐ la variable `x` vaut 5 et la variable `y` vaut 0
17. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :
- ☐ `int factorielle();`
  - ☐ `struct int factorielle(int n);`
  - ☐ `int factorielle(int x);`
  - ☐ `int factorielle(double n);`
18. Après exécution jusqu'à la ligne 15 du programme C :
- ```
10  ...
11  int main() {
12      int x = 5;
13
14      x = 3 * x + 1;
15
16      ...
17  }
```

- ☐ le programme affiche `x`
  - ☐ la variable `x` vaut 16
  - ☐ le programme affiche `****`
  - ☐ la variable `x` vaut  $-\frac{1}{2}$
19. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction
- ☐ `loop i;`
  - ☐ `int loop n;`
  - ☐ `int k;`
  - ☐ `int %d;`
20. Un programme en langage C doit comporter une et une seule définition de la fonction :
- ☐ `init`
  - ☐ `include`
  - ☐ `main`
  - ☐ `begin`

**Barème : 1 point par réponse juste (unique) ; –0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer une fonction **factorielle** qui prend en argument un entier et renvoie sa factorielle on écrit :

☐ `struct int factorielle(int n);`  
☐ `int factorielle();`  
☐ `int factorielle(int x);`  
☐ `int factorielle(double n);`

2. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

☐ `j = 0`  
☐ `j = 4`  
☐ `j = %d`  
☐ `j = 5`

3. Pour déclarer une procédure **afficher\_date** qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

☐ `struct date_s afficher_date(struct date_s d);`  
☐ `void afficher_date(struct date_s d);`  
☐ `int afficher_date(date_s d);`  
☐ `void afficher_date(date_s d);`

4. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

☐ une accolade en trop  
☐ un point-virgule en trop  
☐ une accolade manquante  
☐ un point-virgule manquant

5. Pour déclarer une fonction **saisie\_utilisateur** qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

☐ `int saisie_utilisateur();`  
☐ `saisie_utilisateur(scanf("%d"));`  
☐ `void saisie_utilisateur(int n);`  
☐ `void saisie_utilisateur(char c);`

6. Un fichier source est :

☐ un document illisible pour les humains  
☐ un document de référence du système  
☐ un document qui doit être protégé  
☐ un fichier que l'on doit citer dans les documents produits sur l'ordinateur  
☐ un fichier texte qui sera traduit en instructions processeur

7. Si **factorielle** est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

☐ `printf("%d", factorielle(n));`  
☐ `int factorielle(int 2);`  
☐ `n = factorielle();`  
☐ `n = factorielle(p, q);`

8. Lorsqu'un programme utilise **printf** ou **scanf** il faut qu'il contienne l'instruction préprocesseur :

☐ `#include <stdio.h>`  
☐ `#appart <stdlib.h>`  
☐ `#include <studlib.h>`  
☐ `#include <studio.h>`

9. L'ordonnancement par tourniquet permet :

☐ de ne pas perdre de temps avec la commutation de contexte  
☐ d'afficher des ronds colorés à l'écran  
☐ d'entretenir l'illusion que les processus tournent en parallèle  
☐ de doubler la mémoire disponible

10. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

☐ `x = mccarthy(n);`  
☐ `n = mccarthy(p, q);`  
☐ `n = mccarthy();`  
☐ `int mccarthy(int 2);`

11. Pour compiler un programme **prog.c**, on utilise la ligne de commande :

☐ `gcc -Wall prog.exe -o prog.c`  
☐ `gcc prog.c -o -Wall prog.exe`  
☐ `gcc prog.exe -Wall -o prog.c`  
☐ `gcc -Wall prog.c -o prog.exe`

12. Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :

☐ `int afficher_menu();`  
☐ `double afficher_menu();`  
☐ `int afficher_menu(int char);`  
☐ `char afficher_menu(printf("menu"));`  
☐ `void afficher_menu();`

13. Soit la fonction **g** définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

☐ 7  
☐ 5  
☐ 0

14. L'écriture 111 en binaire correspond au nombre naturel :

☐ 3  
☐ 8  
☐ 111  
☐ 7

15. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il comporte une boucle infinie

16. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42
- ☐ 1
- ☐ 0
- ☐ 6

17. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse syntaxique
- ☐ analyse lexicale
- ☐ analyse sémantique

18. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

19. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
```

```
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = 5
- ☐ j = 0
- ☐ j = 4
- ☐ j = %d

20. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses cases
- ☐ ses blocs
- ☐ ses chants

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`  
☐ `x = racine(2/3);`  
☐ `x - 1 = racine(x);`  
☐ `x = racine(racine(x)*racine(x));`

2. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui ne sera pas détectée  
☐ ne comportent aucune erreur  
☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique  
☐ comportent une erreur qui sera détectée au cours de l'édition de lien

3. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien  
☐ Mineur  
☐ Majeur  
☐ Mineur  
☐ Majeur

4. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant  
☐ un point-virgule en trop  
☐ une accolade manquante  
☐ une accolade en trop

5. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie  
☐ une fonction appelée avant sa déclaration  
☐ un désaccord entre la déclaration et la définition d'une fonction  
☐ une directive préprocesseur **#include** manquante

6. Pour déclarer une fonction **saisie\_utilisateur** qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf("%d"));`  
☐ `void saisie_utilisateur(int n);`  
☐ `void saisie_utilisateur(char c);`  
☐ `int saisie_utilisateur();`

7. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 4  
☐ 0  
☐ 8  
☐ 16

8. Une variable booléenne est un variable :

- ☐ jamais nulle  
☐ à laquelle une valeur vient d'être affectée  
☐ qui est vraie ou fausse  
☐ réelle positive  
☐ NaN (not a number, qui n'est pas un nombre)

9. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4  
☐ 4 3 2 1  
☐ 4 3 2 1 0  
☐ 1 2 3 4

10. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ la variable **x** vaut 16  
☐ le programme affiche **x**  
☐ la variable **x** vaut  $-\frac{1}{2}$   
☐ le programme affiche **\*\*\*\***

11. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`  
☐ `for(i=1;i<=5;i=i+1)`  
☐ `for(i=0;i<5;i=i+1)`  
☐ `for(i=0;i<=5;i=i+1)`

12. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :
- ☐ `n = carre(int n);`
  - ☐ `n = carre(n);`
  - ☐ `int carre(2);`
  - ☐ `int n = carre();`
13. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
- ☐ `printf("Valeur de n ? %g\n", n);`
  - ☐ un débogueur
  - ☐ `printf("Valeur de n ? %d\n", n);`
  - ☐ `scanf("%d", &n);`
14. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses champs
  - ☐ ses chants
  - ☐ ses cases
  - ☐ ses blocs
15. Sous unix (ou linux), la commande `cd` permet de :
- ☐ détruire un fichier
  - ☐ jouer de la musique

- ☐ changer de répertoire courant
  - ☐ ouvrir un bureau partagé (common desktop)
  - ☐ récupérer un programme arrêté avec la commande `ab`
16. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ alphabétique
  - ☐ dans lequel ces fonctions sont appelées dans le `main`
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ un ordre quelconque
17. Le bus système sert à :
- ☐ Arriver à l'heure en cours
  - ☐ transporter les processus du tourniquet au processeur
  - ☐ Transférer des données et intructions entre processeur et mémoire
  - ☐ Écrire des données sur le disque dur
18. Au début de la fonction `main()` on place le code :
- ```
char i;  
for (i = 'A'; i <= 'F'; i = i + 1)  
{
```

- ```
    printf("%c", i);  
}  
printf("\n");
```
- Alors l'affichage sera :
- ☐ ABCDEF
  - ☐ i
  - ☐ A
  - ☐ cccccc
19. Vous utilisez une boucle `while` quand :
- ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ l'incrément de la variable de boucle n'est pas 1
20. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :
- ☐ `int pgcd(int y, int x);`
  - ☐ `int pgcd(int x, int x);`
  - ☐ `int pgcd(int x, y);`
  - ☐ `void pgcd(int x, int y);`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur  
Majeur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

2. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(x * x) - racine(x);`

3. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ `loop i;`
- ☐ `int k;`
- ☐ `int loop n;`
- ☐ `int %d;`

4. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique

- ☐ dans lequel ces fonctions sont appelées dans le `main`
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque

5. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu(int char);`
- ☐ `void afficher_menu();`
- ☐ `double afficher_menu();`
- ☐ `char afficher_menu printf("menu");`
- ☐ `int afficher_menu();`

6. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define taille = N`
- ☐ `#define N 3`
- ☐ `#define taille = 3`
- ☐ `#define N = 3`

7. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a == b`
- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`

8. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%x et y=%y\n");`

9. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ le bus explose
- ☐ la case mémoire 8 contiendra 16
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0

10. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `char "c";`

11. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui ne sera pas détectée

12. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il n'affiche rien

13. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 2 4 6 8
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6
- ☐ 0 1 2 3 4 5 6

14. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4
- ☐ 1
- ☐ 5
- ☐ 0

15. Après exécution jusqu'à la ligne 14 du programme C :

```
10  int main() {
11      int x = 5;
12
13      printf(" x = %d\n", 2);
14
15      ...
16  }
```

- ☐ le terminal affiche `x = 5`
- ☐ le terminal affiche 5
- ☐ le terminal affiche `x = 2`
- ☐ le terminal affiche "Faux"

16. Un bit est :

- ☐ l'instruction qui met fin à un programme
- ☐ la longueur d'un mot mémoire
- ☐ un battement d'horloge processeur
- ☐ un chiffre binaire (0 ou 1)

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur  
Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

18. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `struct toto_s toto;`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `int toto.n = 3;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `toto_s n, x;`

19. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0

20. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `void exposant(double x^n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ un point-virgule manquant
- ☐ une accolade manquante
- ☐ un point-virgule en trop
- ☐ une accolade en trop

2. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale
- ☐ analyse harmonique
- ☐ analyse syntaxique
- ☐ analyse sémantique

3. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui ne sera pas détectée
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique

4. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `int char;`
- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `char c;`

5. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`
- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`

6. On considère deux variables booléennes A et B initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?

- ☐ `(!A || B)`
- ☐ `A && B`
- ☐ `(A == TRUE) && (B == TRUE)`
- ☐ `!(A || B) == (A && !B)`

7. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 0
- ☐ 1
- ☐ 5
- ☐ 4

8. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

9. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n",x y);`

10. Si `x` est une variable réelle (de type `double`) alors `x = 3/2` lui affecte la valeur :

- ☐ 1.5
- ☐ 0
- ☐ 0.5
- ☐ 1

11. Si cette erreur apparaît à la compilation :

**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ une variable non déclarée
- ☐ une directive préprocesseur `#include` manquante
- ☐ un caractère interdit en C
- ☐ une faute de frappe dans un appel de fonction

12. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur scanf(%d);`
- ☐ `void saisie_utilisateur(char c);`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(int n);`

13. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(2/3);`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x - 1 = racine(x);`

14. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7
- ☐ 0
- ☐ 5

15. Le type des réels en C est :

- ☐ `double`
- ☐ `real`
- ☐ `int`
- ☐ `char`

16. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le `main`
- ☐ un ordre quelconque
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique

17. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur  
Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ rien

18. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction appelée avant sa déclaration

19. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `n = pgcd(int p, int q);`
- ☐ `int pgcd(2);`
- ☐ `n = pgcd(n, 3);`
- ☐ `int n = pgcd();`

20. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int toto[5];`
- ☐ `int toto[taille=5];`
- ☐ `char tableau[5];`
- ☐ `int[] new tableau(5);`
- ☐ `int tab[] = 5;`

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 1 2 3 4
- ☐ 0 1 2 3 4

2. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ l'incrément de la variable de boucle n'est pas 1

3. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche `x = 2`
- ☐ le terminal affiche `x = 5`
- ☐ le terminal affiche 5
- ☐ le terminal affiche "Faux"

4. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse lexicale
- ☐ analyse sémantique
- ☐ analyse syntaxique
- ☐ analyse harmonique

5. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 5
- ☐ 101
- ☐ 4
- ☐ 3

6. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x = racine(2/3);`

7. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=13, a=8, b=5`

8. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou

9. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ mettre les blocs en séquence les uns à la suite des autres
- ☐ retourner un bloc
- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ répéter un bloc tant qu'une condition est vérifiée

10. Un registre du processeur est :

- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

11. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

12. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char "c";`

13. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`
- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`
- ☐ `void exposant(double x^n);`

14. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ des processus
- ☐ les fichiers du disque
- ☐ certaines données de la mémoire de travail
- ☐ en temps d'accès

15. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 4
- ☐ 5
- ☐ 0
- ☐ 1

16. Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 8 2
- ☐ 8 6 4 2 0
- ☐ 0 2 4 6 8
- ☐ 8 6 4 2

17. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :  
**Undefined symbols : "\_printf" ou**  
**référence indéfinie vers « printf »**

- ☐ l'édition de liens
- ☐ l'analyse des entrées clavier
- ☐ l'analyse harmonique
- ☐ l'analyse sémantique

18. Au début de la fonction **main()** on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ B
- ☐ C
- ☐ A
- ☐ b

19. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy(p, q);`
- ☐ `n = mccarthy();`
- ☐ `x = mccarthy(n);`

20. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ comportent une erreur qui ne sera pas détectée

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui ne sera pas détectée

2. L'ordonnancement par tourniquet permet :

- ☐ de doubler la mémoire disponible
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'afficher des ronds colorés à l'écran
- ☐ d'entretenir l'illusion que les processus tournent en parallèle

3. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 111
- ☐ 7
- ☐ 8
- ☐ 3

4. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses champs
- ☐ ses chants
- ☐ ses blocs
- ☐ ses cases

5. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 15
- ☐ 10
- ☐ 6
- ☐ 0

6. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur
- ☐ que l'on veut voir tous les avertissements

7. Le code suivant :

```
int i;
for (i = 1; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 1 2 3 4
- ☐ 0 1 2 3 4
- ☐ 4 3 2 1 0
- ☐ 4 3 2 1

8. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(x * x) - racine(x);`

9. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char 'c';`

10. Le langage C est un langage

- ☐ compilé
- ☐ composé
- ☐ interprété
- ☐ lu, écrit, parlé

11. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`

12. Pour compiler un programme `prog.c`, on utilise la ligne de commande :

- ☐ `gcc prog.exe -Wall -o prog.c`
- ☐ `gcc -Wall prog.c -o prog.exe`
- ☐ `gcc -Wall prog.exe -o prog.c`
- ☐ `gcc prog.c -o -Wall prog.exe`

13. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :
- ☐ `n = carre(n);`
  - ☐ `int n = carre();`
  - ☐ `n = carre(int n);`
  - ☐ `int carre(2);`
14. L'écriture 101 en binaire correspond au nombre naturel :
- ☐ 101
  - ☐ 3
  - ☐ 5
  - ☐ 4
15. Avant de faire appel à une fonction il est nécessaire de :
- ☐ avoir défini une constante symbolique de la taille de cette fonction
  - ☐ l'avoir déclarée et définie
  - ☐ l'avoir définie
  - ☐ l'avoir déclarée
16. Si cette erreur apparaît à la compilation :  
**error: expected ';' before '}' token** que doit-on chercher dans le programme ?
- ☐ une accolade manquante

- ☐ une accolade en trop
  - ☐ un point-virgule en trop
  - ☐ un point-virgule manquant
17. Soit un programme contenant les lignes suivantes :
- ```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```
- qu'est ce qui sera affiché ?
- ☐ `j = 0`
  - ☐ `j = %d`
  - ☐ `j = 5`
  - ☐ `j = 4`
18. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
- ☐ `exposant(double x, int n, int r);`

- ☐ `double exposant(double x, int n);`
  - ☐ `void exposant(double x^n);`
  - ☐ `int exposant(double n, int x);`
19. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :
- ☐ `int afficher_menu();`
  - ☐ `double afficher_menu();`
  - ☐ `int afficher_menu(int char);`
  - ☐ `char afficher_menu(sprintf("menu"));`
  - ☐ `void afficher_menu();`
20. Au début de la fonction `main()` on place le code :
- ```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```
- Alors l'affichage sera :
- ☐ ABCDEF
  - ☐ i
  - ☐ A
  - ☐ cccccc

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y = 3;
13
14      x = y;
15
16      ...
17  }
```

- ☐ la variable x vaut 5 et la variable y vaut 3
- ☐ la variable x vaut 3
- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"

2. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

3. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `char afficher_menu(printf("menu"));`
- ☐ `double afficher_menu();`
- ☐ `int afficher_menu(int char);`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu();`

4. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ `a == b`
- ☐ `a = b`
- ☐ `a{n, x} == b{n, x}`
- ☐ `(a.n == b.n) && (a.x == b.x)`

5. Le type des réels en C est :

- ☐ `char`
- ☐ `double`
- ☐ `real`
- ☐ `int`

6. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char c;`
- ☐ `char "c";`

7. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

8. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ b
- ☐ A
- ☐ B
- ☐ C

9. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction

10. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse syntaxique
- ☐ analyse sémantique
- ☐ analyse lexicale
- ☐ analyse harmonique

11. Un bit est :

- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)
- ☐ la longueur d'un mot mémoire
- ☐ un battement d'horloge processeur

12. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `double exposant(double x, int n);`
- ☐ `int exposant(double n, int x);`
- ☐ `void exposant(double x^n);`
- ☐ `exposant(double x, int n, int r);`

13. Soient deux variables entières  $x$  et  $y$  initialisées à 4 et 5 respectivement. L'affichage  $x=4$  et  $y=5$  est obtenu avec la commande :

- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x,y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%d et y=%d\n",x y);`

14. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses blocs
- ☐ ses champs
- ☐ ses cases

15. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 5
- ☐ j = %d
- ☐ j = 4
- ☐ j = 0

16. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ un débogueur
- ☐ `printf("Valeur de n ? %g\n", n);`

17. Quel est l'opérateur de différence en C :

- ☐ `!=`
- ☐ `!`
- ☐ `≠`
- ☐ `<>`

18. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 3
- ☐ 16
- ☐ 20

19. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studlib.h>`
- ☐ `#include <studio.h>`

20. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 0
- ☐ 10
- ☐ 15



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses chants
- ☐ ses blocs
- ☐ ses champs
- ☐ ses cases

2. Un fichier source est :

- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document illisible pour les humains
- ☐ un document qui doit être protégé
- ☐ un document de référence du système
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur

3. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ des processus
- ☐ certaines données de la mémoire de travail
- ☐ les fichiers du disque
- ☐ en temps d'accès

4. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 4
- ☐ 0
- ☐ 16
- ☐ 8

5. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1

6. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 3 0 1 2
- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 0 1 2 3

7. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4
- ☐ 1
- ☐ 0
- ☐ 5

8. Une variable booléenne est un variable :

- ☐ jamais nulle
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ réelle positive
- ☐ à laquelle une valeur vient d'être affectée
- ☐ qui est vraie ou fausse

9. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Mineur  
Majeur
- ☐ rien

10. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x,y);`

11. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int toto[taille=5];`
- ☐ `int tab[] = 5;`
- ☐ `int toto[5];`
- ☐ `char tableau[5];`
- ☐ `int[] new tableau(5);`

12. L'écriture 111 en binaire correspond au nombre naturel :
- ☐ 8
  - ☐ 3
  - ☐ 7
  - ☐ 111
13. Un programme en langage C doit comporter une et une seule définition de la fonction :
- ☐ main
  - ☐ begin
  - ☐ init
  - ☐ include
14. Quel est le problème d'un programme comportant les lignes suivantes ?
- ```
while (1)
{
    printf("coucou\n");
}
```
- ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il comporte une boucle infinie
  - ☐ il ne compile pas
  - ☐ il n'affiche rien

15. Le bus système sert à :
- ☐ Transférer des données et intructions entre processeur et mémoire
  - ☐ transporter les processus du tourniquet au processeur
  - ☐ Écrire des données sur le disque dur
  - ☐ Arriver à l'heure en cours
16. Au début de la fonction `main()` on place le code :
- ```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```
- Alors l'affichage sera :
- ☐ b
  - ☐ A
  - ☐ C
  - ☐ B
17. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :
- ☐ `int carre(2);`
  - ☐ `n = carre(int n);`
  - ☐ `n = carre(n);`
  - ☐ `int n = carre();`

18. L'écriture 101 en binaire correspond au nombre naturel :
- ☐ 5
  - ☐ 4
  - ☐ 3
  - ☐ 101
19. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :
- ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x = racine(2/3);`
  - ☐ `x = racine(x * x) - racine(x);`
  - ☐ `x - 1 = racine(x);`
20. Au début de la fonction `main()` on place le code :
- ```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```
- Alors l'affichage sera :
- ☐ A
  - ☐ i
  - ☐ cccccc
  - ☐ ABCDEF

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

☐ `int toto[taille=5];`  
☐ `int tab[] = 5;`  
☐ `int[] new tableau(5);`  
☐ `int toto[5];`  
☐ `char tableau[5];`

2. Une *segmentation fault* est une erreur qui survient lorsque :

☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal  
☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée  
☐ la division du programme en zones homogènes échoue  
☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur

3. Au début de la fonction `main()` on place le code :

```
char i;  
for (i = 'A'; i <= 'F'; i = i + 1)  
{  
    printf("%c", i);  
}  
printf("\n");
```

Alors l'affichage sera :

☐ A  
☐ i  
☐ ABCDEF  
☐ cccccc

4. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

☐ `scanf("%d", &n);`  
☐ `printf("Valeur de n ? %g\n", n);`  
☐ un débogueur  
☐ `printf("Valeur de n ? %d\n", n);`

5. Soit la fonction `g` définie par :

```
int g(int a)  
{  
    printf("a = \n", %d);  
    if (1 > 0)  
    {  
        return 5;  
    }  
    return 7;  
}
```

Alors l'expression `g(0)` prendra la valeur :

☐ 5  
☐ 7  
☐ 0

6. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

☐ `#include <stdlib.h>`  
☐ `#appart <stdlib.h>`  
☐ `#include <stdio.h>`  
☐ `#include <studio.h>`

7. Le code suivant :

```
int age = 18;  
if (age < 18)  
{  
    printf("Mineur\n");  
}  
else  
{  
    printf("Majeur\n");  
}
```

affichera :

☐ Mineur  
☐ Mineur  
Majeur  
☐ Majeur  
☐ rien

8. L'écriture 101 en binaire correspond au nombre naturel :

☐ 3  
☐ 5  
☐ 101  
☐ 4

9. Le code suivant :

```
int i;  
for (i = 4; i >= 0; i = i - 1)  
{  
    printf("%d ", i);  
}  
printf("\n");
```

affichera :

☐ 1 2 3 4  
☐ 4 3 2 1 0  
☐ 4 3 2 1  
☐ 0 1 2 3 4

10. Le type des réels en C est :

☐ `double`  
☐ `int`  
☐ `real`  
☐ `char`

11. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {  
11     int x = 5;  
12     int y = 3;  
13  
14     x = y;  
15  
16     ...  
17 }
```

☐ la variable `y` vaut 5  
☐ la variable `x` vaut 3  
☐ le programme affiche "Faux"  
☐ la variable `x` vaut 5 et la variable `y` vaut 3

12. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 0
- ☐ j = 4
- ☐ j = %d
- ☐ j = 5

13. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque
- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main

14. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il ne compile pas
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou

15. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ char 'c';
- ☐ char "c";
- ☐ char c;
- ☐ int char;

16. On souhaite faire une boucle de contrôle de saisie : tant que l'entier n n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de n. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition cond :

- ☐ (n<=a) && (n<=b)
- ☐ (a<n) || (n>b)
- ☐ (a<=n) && (n<=b)
- ☐ a<=n<=b

17. Une variable booléenne est une variable :

- ☐ à laquelle une valeur vient d'être affectée
- ☐ réelle positive
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ jamais nulle
- ☐ qui est vraie ou fausse

18. Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 42
- ☐ 6
- ☐ 1
- ☐ 0

19. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse syntaxique
- ☐ analyse harmonique
- ☐ analyse lexicale
- ☐ analyse sémantique

20. Vous utilisez une boucle while quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un for dans le même programme principal
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :
  - ☐ `x = racine(2/3);`
  - ☐ `x = racine(x * x) - racine(x);`
  - ☐ `x = racine(racine(x)*racine(x));`
  - ☐ `x - 1 = racine(x);`
- Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :
  - ☐ `void afficher_menu();`
  - ☐ `char afficher_menu printf("menu");`
  - ☐ `int afficher_menu();`
  - ☐ `int afficher_menu(int char);`
  - ☐ `double afficher_menu();`
- Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`, que doit-on chercher dans le programme ?
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction appelée avant sa déclaration
- Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :
  - ☐ analyse syntaxique
  - ☐ analyse harmonique
  - ☐ analyse sémantique
  - ☐ analyse lexicale
- Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il comporte une boucle infinie
  - ☐ il n'affiche rien
- Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :
    - ☐ 0 1 2 3 4
    - ☐ 4 3 2 1 0
    - ☐ 4 3 2 1
    - ☐ 1 2 3 4
  - Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :
    - ☐ `printf("x=%d et y=%d\n",x,y);`
    - ☐ `printf("x=%d et y=%d\n",x y);`
    - ☐ `printf("x=%d et y=%d\n,x,y");`
    - ☐ `printf("x=%x et y=%y\n");`
  - Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :
    - ☐ C
    - ☐ B
    - ☐ A
    - ☐ b

- Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression `f(1)` prendra la valeur :

- ☐ 4
  - ☐ 0
  - ☐ 5
  - ☐ 1
- Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :
    - ☐ tous ensemble
    - ☐ chacun son tour, après que le processus précédent a terminé
    - ☐ tour à tour, un petit peu à chaque fois
    - ☐ en parallèle, chacun dans un registre
  - Le langage C est un langage
    - ☐ lu, écrit, parlé
    - ☐ composé
    - ☐ interprété
    - ☐ compilé
  - Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=13, a=8, b=5

13. Le type des réels en C est :

- ☐ int
- ☐ char
- ☐ double
- ☐ real

14. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <studio.h>`
- ☐ `#include <studlib.h>`
- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`

15. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3
- ☐ 20
- ☐ 16
- ☐ 6

16. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ qu'il faut lancer un débogueur

17. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ changer de répertoire courant
- ☐ créer un fichier texte
- ☐ ouvrir un fichier texte
- ☐ créer un répertoire

18. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int toto[5];`
- ☐ `int tab[] = 5;`
- ☐ `int[] new tableau(5);`
- ☐ `char tableau[5];`
- ☐ `int toto[taille=5];`

19. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int carre(2);`
- ☐ `int n = carre();`
- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`

20. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.

☐ #define taille = 3  
☐ #define N 3  
☐ #define N = 3  
☐ #define taille = N

2. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

☐ char "c";  
☐ char c;  
☐ char 'c';  
☐ int char;

3. Pour déclarer une fonction pgcd qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

☐ int pgcd(int x, y);  
☐ int pgcd(int y, int x);  
☐ int pgcd(int x, int x);  
☐ void pgcd(int x, int y);

4. Un enregistrement permet de grouper plusieurs valeurs dans :

☐ ses champs  
☐ ses cases  
☐ ses chants  
☐ ses blocs

5. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

☐ for(i=0;i<5;i=i+1)  
☐ for(i=1;i<=5;i=i+1)  
☐ for(i=0;i<=5;i=i+1)  
☐ for(i=1;i<5;i=i+1)

6. Si factorielle est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

☐ n = factorielle(p, q);  
☐ printf("%d", factorielle(n));  
☐ int factorielle(int 2);  
☐ n = factorielle();

7. Si pgcd est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

☐ int pgcd(2);  
☐ n = pgcd(n, 3);  
☐ n = pgcd(int p, int q);  
☐ int n = pgcd();

8. Au début de la fonction main() on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

☐ B  
☐ A  
☐ C  
☐ b

9. Si carre est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que n est une variable entière définie et initialisée, il est correct d'écrire :

☐ n = carre(int n);  
☐ n = carre(n);  
☐ int n = carre();  
☐ int carre(2);

10. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
}
```

```
}
printf(" ");
}
```

qu'est ce qui sera affiché ?

☐ \*\* \*\* \*\* \*\*  
☐ \*\*\*\* \*\*  
☐ \*\*\*\*\*  
☐ \*\* \*\* \*\*

11. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

☐ 3  
☐ 0  
☐ 4

12. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

☐ 16  
☐ 20  
☐ 6  
☐ 3

13. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ alphabétique
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ un ordre quelconque
  - ☐ dans lequel ces fonctions sont appelées dans le main
14. Sous unix (ou linux), la commande `ls` permet de :
- ☐ compiler un programme
  - ☐ voir des clips musicaux
  - ☐ afficher le contenu d'un fichier texte
  - ☐ afficher la liste de fichiers contenus dans un répertoire
15. Le langage C est un langage
- ☐ compilé
  - ☐ interprété
  - ☐ lu, écrit, parlé
  - ☐ composé

16. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :
- ☐ `#include <studio.h>`
  - ☐ `#include <studlib.h>`
  - ☐ `#appart <stdlib.h>`
  - ☐ `#include <stdio.h>`
17. Le code suivant :
- ```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```
- affichera :
- ☐ Mineur
  - ☐ Majeur
  - ☐ rien
  - ☐ Majeur
  - ☐ Mineur

18. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :
- ☐ `int mccarthy(int 2);`
  - ☐ `x = mccarthy(n);`
  - ☐ `n = mccarthy(p, q);`
  - ☐ `n = mccarthy();`
19. Pour déclarer une fonction **exposant** qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :
- ☐ `int exposant(double n, int x);`
  - ☐ `double exposant(double x, int n);`
  - ☐ `void exposant(double x^n);`
  - ☐ `exposant(double x, int n, int r);`
20. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?
- ☐ une fonction appelée avant sa déclaration
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction



**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6 8
- ☐ 0 1 2 3 4 5 6
- ☐ 0 2 4 6

2. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou

3. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ certaines données de la mémoire de travail
- ☐ en temps d'accès
- ☐ des processus
- ☐ les fichiers du disque

4. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut lancer un débogueur
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut indenter le fichier source

5. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 111
- ☐ 8
- ☐ 7

6. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ un débogueur
- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ `scanf("%d", &n);`
- ☐ `printf("Valeur de n ? %g\n", n);`

7. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=1;i<5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`

8. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration

9. Si  $x$  est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

- ☐ 1
- ☐ 0
- ☐ 1.5
- ☐ 0.5

10. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

11. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 5
- ☐ 3
- ☐ 4

12. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée et définie
- ☐ l'avoir définie
- ☐ l'avoir déclarée
- ☐ avoir défini une constante symbolique de la taille de cette fonction

13. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle(p, q);`
- ☐ `n = factorielle();`
- ☐ `int factorielle(int 2);`
- ☐ `printf("%d", factorielle(n));`

14. Un fichier source est :

- ☐ un document qui doit être protégé
- ☐ un fichier texte qui sera traduit en instructions processeur
- ☐ un document de référence du système
- ☐ un document illisible pour les humains
- ☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur

15. Le langage C est un langage

- ☐ interprété
- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ composé

16. Si cet avertissement apparaît à la compilation :

**warning: implicit declaration of function 'max', que doit-on chercher dans le programme?**

- ☐ une fonction déclarée mais non définie
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur **#include** manquante

17. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
```

```
printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=8, a=3, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=13, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5

18. Si pgcd est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ n = pgcd(int p, int q);
- ☐ n = pgcd(n, 3);
- ☐ int n = pgcd();
- ☐ int pgcd(2);

19. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
```

```
15
16      ...
17  }
```

- ☐ la variable x vaut 5 et la variable y vaut 0
- ☐ la variable x vaut 0
- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"

20. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = %d
- ☐ j = 5
- ☐ j = 0
- ☐ j = 4

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 4
- ☐ j = 5
- ☐ j = 0
- ☐ j = %d

2. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle(p, q);`
- ☐ `n = factorielle();`
- ☐ `printf("%d", factorielle(n));`
- ☐ `int factorielle(int 2);`

3. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

4. Avant de faire appel à une fonction il est nécessaire de :

- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir déclarée
- ☐ l'avoir déclarée et définie
- ☐ l'avoir définie

5. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 4
- ☐ 0
- ☐ 8

6. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tous ensemble
- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ en parallèle, chacun dans un registre
- ☐ tour à tour, un petit peu à chaque fois

7. Quels calculs peut-on programmer en programmation structurée ?

- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine

8. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `begin`
- ☐ `init`
- ☐ `include`
- ☐ `main`

9. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ B
- ☐ b
- ☐ A
- ☐ C

10. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut lancer un débogueur

11. Si cette erreur apparaît à la compilation : **error: expected ';' before '}' token** que doit-on chercher dans le programme ?

- ☐ une accolade manquante
- ☐ un point-virgule manquant
- ☐ une accolade en trop
- ☐ un point-virgule en trop

12. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Majeur
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur

13. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int toto[taille=5];`
- ☐ `int[] new tableau(5);`
- ☐ `int tab[] = 5;`
- ☐ `char tableau[5];`
- ☐ `int toto[5];`

14. Après la déclaration : `int mccarthy(int n);`, il est correct d'écrire :

- ☐ `n = mccarthy(p, q);`
- ☐ `x = mccarthy(n);`
- ☐ `int mccarthy(int 2);`
- ☐ `n = mccarthy();`

15. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int pgcd(2);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `n = pgcd(n, 3);`
- ☐ `int n = pgcd();`

16. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ alphabétique
- ☐ un ordre quelconque

17. Une variable booléenne est un variable :

- ☐ à laquelle une valeur vient d'être affectée
- ☐ réelle positive
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ jamais nulle
- ☐ qui est vraie ou fausse

18. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
```

```
{
    printf("%d ", j);
}
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 3 0 1 2

19. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `char c;`
- ☐ `int char;`

20. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int x, int x);`
- ☐ `void pgcd(int x, int y);`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur  
Majeur

2. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ kwrite TP4
- ☐ mkdir TP4
- ☐ yppasswd
- ☐ new TP4

3. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

4. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses cases
- ☐ ses champs
- ☐ ses chants

5. On souhaite faire une boucle de contrôle de saisie : tant que l'entier n n'appartient pas à l'intervalle [a..b], on recommence la saisie de n. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition cond :

- ☐ (a<=n) && (n<=b)
- ☐ a<=n<=b
- ☐ (a<n) || (n>b)
- ☐ (n<=a) && (n<=b)

6. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse syntaxique
- ☐ analyse lexicale
- ☐ analyse sémantique
- ☐ analyse harmonique

7. Un registre du processeur est :

- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs
- ☐ un composant qui contient la liste des fichiers du système
- ☐ une unité de calcul spécialisée de l'ordinateur

8. Lorsqu'un programme utilise printf ou scanf il faut qu'il contienne l'instruction préprocesseur :

- ☐ #appart <stdlib.h>
- ☐ #include <studlib.h>
- ☐ #include <studio.h>
- ☐ #include <stdio.h>

9. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 0 1 2
- ☐ 0 1 2 3 0 1 2

10. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 0
- ☐ 5
- ☐ 1
- ☐ 4

11. Si a et b sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de a et de b on utilise la condition :

- ☐ a = b
- ☐ a{n, x} == b{n, x}
- ☐ a == b
- ☐ (a.n == b.n) && (a.x == b.x)

12. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 16
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0
- ☐ le bus explose

13. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 0 1 1 1
- ☐ 0 1 2 0 1 2
- ☐ 1 2 1 2 3
- ☐ 0 1 0 1 0 1 0 1

14. Soit la fonction g définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression g(0) prendra la valeur :

- ☐ 0
- ☐ 7
- ☐ 5

15. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

```
☐ struct date_s afficher_date(struct date_s d);
☐ int afficher_date(date_s d);
☐ void afficher_date(date_s d);
☐ void afficher_date(struct date_s d);
```

16. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1

17. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir définie
- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir déclarée et définie
- ☐ l'avoir déclarée

18. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il n'affiche rien
- ☐ il risque d'afficher bonjour à la place de coucou

19. Dans la commande gcc, l'option `-Wall` signifie :

- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur
- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements

20. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*
- ☐ \*\* \*\*\* \*\*\*\*\*
- ☐ \*\*\*\*\* \*\*
- ☐ \*\*\*\* \*\*

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Après exécution du programme :

```
1  lecture 8 r0
2  valeur 3 r1
3  mult r1 r0
4  valeur 1 r2
5  add r2 r0
6  ecriture r0 8
7  stop
8  5
```

- ☐ la case mémoire 8 contiendra 16
- ☐ le bus explose
- ☐ le terminal affiche 8
- ☐ la case mémoire 8 contiendra 0

2. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur
- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée

3. Le langage C est un langage

- ☐ composé
- ☐ interprété
- ☐ lu, écrit, parlé
- ☐ compilé

4. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ alphabétique
- ☐ dans lequel vous avez déclaré ces fonction

☐ dans lequel ces fonctions sont appelées dans le main

☐ un ordre quelconque

5. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il n'affiche rien
- ☐ il comporte une boucle infinie
- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il ne compile pas

6. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`
- ☐ `n = pgcd(n, 3);`
- ☐ `n = pgcd(int p, int q);`
- ☐ `int pgcd(2);`

7. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un répertoire
- ☐ créer un fichier texte
- ☐ changer de répertoire courant
- ☐ ouvrir un fichier texte

8. Une variable booléenne est un variable :

- ☐ jamais nulle
- ☐ à laquelle une valeur vient d'être affectée
- ☐ réelle positive
- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ qui est vraie ou fausse

9. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf ?

- ☐ j = 0
- ☐ j = %d
- ☐ j = 5
- ☐ j = 4

10. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ les fichiers du disque
- ☐ des processus
- ☐ certaines données de la mémoire de travail
- ☐ en temps d'accès

11. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?

- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char "c";`

12. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `int factorielle(int 2);`
- ☐ `printf("%d", factorielle(n));`
- ☐ `n = factorielle();`
- ☐ `n = factorielle(p, q);`

13. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ Majeur
- ☐ Mineur
- ☐ rien

14. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ mkdir TP4
- ☐ new TP4
- ☐ kwrite TP4
- ☐ yppasswd

15. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`
- ☐ `void afficher_date(struct date_s d);`

16. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `int exposant(double n, int x);`
- ☐ `double exposant(double x, int n);`
- ☐ `void exposant(double x^n);`
- ☐ `exposant(double x, int n, int r);`

17. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie
- ☐ une directive préprocesseur `#include` manquante

18. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
    printf("\n");
}
```

qu'est ce qui sera affiché ?

- ☐ 0 0 1 1 2 2
- ☐ 1 2 3 1 2
- ☐ 0 1 2 0 1 2
- ☐ 0 1 0 1 0 1

19. Après exécution jusqu'à la ligne 14 du programme C :

```
10 int main() {
11     int x = 5;
12
13     printf(" x = %d\n", 2);
14
15     ...
16 }
```

- ☐ le terminal affiche `x = 2`
- ☐ le terminal affiche "Faux"
- ☐ le terminal affiche `x = 5`
- ☐ le terminal affiche 5

20. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 4
- ☐ 0



**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. On souhaite faire une boucle de contrôle de saisie : tant que l'entier  $n$  n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de  $n$ . Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`  
☐ `(a<n) || (n>b)`  
☐ `a<=n<=b`  
☐ `(n<=a) && (n<=b)`

2. Si  $a$  et  $b$  sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de  $a$  et de  $b$  on utilise la condition :

- ☐ `a = b`  
☐ `a == b`  
☐ `(a.n == b.n) && (a.x == b.x)`  
☐ `a{n, x} == b{n, x}`

3. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le `main` est le suivant :

- ☐ `f(3,5)=8, a=3, b=5`  
☐ `f(a,b)=13, a=8, b=5`  
☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=8, a=3, b=5`

4. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de `somme` affichée est :

- ☐ 15  
☐ 6  
☐ 0  
☐ 10

5. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée  
☐ l'avoir déclarée et définie  
☐ l'avoir définie  
☐ avoir défini une constante symbolique de la taille de cette fonction

6. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ ne comportent aucune erreur

- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique  
☐ comportent une erreur qui ne sera pas détectée  
☐ comportent une erreur qui sera détectée au cours de l'édition de lien

7. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur  
☐ Mineur  
     Majeur  
☐ Majeur  
☐ rien

8. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`  
☐ `int pgcd(int y, int x);`  
☐ `int pgcd(int x, int x);`  
☐ `int pgcd(int x, y);`

9. Si `pgcd` est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ `int n = pgcd();`  
☐ `n = pgcd(int p, int q);`  
☐ `n = pgcd(n, 3);`  
☐ `int pgcd(2);`

10. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :
- ☐ `void afficher_date(date_s d);`
  - ☐ `struct date_s afficher_date(struct date_s d);`
  - ☐ `int afficher_date(date_s d);`
  - ☐ `void afficher_date(struct date_s d);`
11. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :
- ☐ chacun son tour, après que le processus précédent a terminé
  - ☐ tour à tour, un petit peu à chaque fois
  - ☐ tous ensemble
  - ☐ en parallèle, chacun dans un registre
12. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?
- ☐ `char 'c';`
  - ☐ `int char;`
  - ☐ `char "c";`
  - ☐ `char c;`
13. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :
- ☐ `printf("x=%d et y=%d\n",x,y);`
  - ☐ `printf("x=%x et y=%y\n");`
  - ☐ `printf("x=%d et y=%d\n",x,y);`
  - ☐ `printf("x=%d et y=%d\n",x y);`
14. Le code suivant :
- ```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ rien
- ☐ Mineur  
Majeur
- ☐ Majeur
- ☐ Mineur

15. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :
- ☐ `for(i=1;i<5;i=i+1)`
  - ☐ `for(i=0;i<=5;i=i+1)`
  - ☐ `for(i=1;i<=5;i=i+1)`
  - ☐ `for(i=0;i<5;i=i+1)`
16. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
- ☐ alphabétique
  - ☐ un ordre quelconque
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ dans lequel ces fonctions sont appelées dans le main
17. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :
- ☐ `int n = carre();`
  - ☐ `n = carre(n);`
  - ☐ `int carre(2);`
  - ☐ `n = carre(int n);`

18. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0
- ☐ 4 3 2 1
- ☐ 0 1 2 3 4
- ☐ 1 2 3 4

19. Quel est le problème d'un programme comportant les lignes suivantes ?
- ```
while (1)
{
    printf("coucou\n");
}
```
- ☐ il n'affiche rien
  - ☐ il comporte une boucle infinie
  - ☐ il risque d'afficher bonjour à la place de coucou
  - ☐ il ne compile pas
20. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :
- ☐ les fichiers du disque
  - ☐ en temps d'accès
  - ☐ des processus
  - ☐ certaines données de la mémoire de travail

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le type des réels en C est :

- ☐ int  
☐ char  
☐ double  
☐ real

2. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3  
☐ 4  
☐ 0

3. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A  
☐ cccccc  
☐ i  
☐ ABCDEF

4. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ un ordre quelconque

- ☐ dans lequel vous avez déclaré ces fonction

- ☐ alphabétique

- ☐ dans lequel ces fonctions sont appelées dans le `main`

5. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur  
☐ Mineur  
Majeur  
☐ Majeur  
☐ rien

6. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs  
☐ ses chants  
☐ ses cases  
☐ ses champs

7. Le bus système sert à :

- ☐ Écrire des données sur le disque dur  
☐ Transférer des données et intructions entre processeur et mémoire  
☐ Arriver à l'heure en cours  
☐ transporter les processus du tourniquet au processeur

8. Soient deux variables entières `x` et `y` initialisées à 4 et 5 respectivement. L'affichage `x=4` et `y=5` est obtenu avec la commande :

- ☐ `printf("x=%d et y=%d\n",x,y);`  
☐ `printf("x=%x et y=%y\n");`  
☐ `printf("x=%d et y=%d\n",x y);`  
☐ `printf("x=%d et y=%d\n",x,y);`

9. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`  
☐ `int exposant(double n, int x);`  
☐ `void exposant(double x^n);`  
☐ `double exposant(double x, int n);`

10. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ Mineur  
☐ Majeur  
☐ Mineur  
Majeur  
☐ rien

11. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche \*\*\*\*  
☐ la variable `x` vaut  $-\frac{1}{2}$   
☐ la variable `x` vaut 16  
☐ le programme affiche `x`

12. Sur un ordinateur avec un seul processeur, habituellement les processus sont exécutés :

- ☐ tour à tour, un petit peu à chaque fois
- ☐ en parallèle, chacun dans un registre
- ☐ chacun son tour, après que le processus précédent a terminé
- ☐ tous ensemble

13. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 2; j = j + 1)
    {
        printf("%d ", i);
    }
}
printf("\n");
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 0 1 0 1 0 1
- ☐ 0 0 1 1 2 2
- ☐ 1 2 3 1 2

14. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 101
- ☐ 5
- ☐ 3
- ☐ 4

15. Pour l'extrait de programme suivant :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", j);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 3 0 1 2
- ☐ 0 0 1 1 2 2 3
- ☐ 0 1 2 0 1 2 3
- ☐ 0 1 2 0 1 2

16. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5
- ☐ f(3,5)=8, a=3, b=5

☐ f(a,b)=8, a=8, b=5

☐ f(a,b)=8, a=3, b=5

17. Le langage C est un langage

- ☐ composé
- ☐ interprété
- ☐ compilé
- ☐ lu, écrit, parlé

18. Pour déclarer une procédure **afficher\_menu** sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu();`
- ☐ `double afficher_menu();`
- ☐ `void afficher_menu();`
- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu(int char);`

19. Lorsqu'un programme utilise **printf** ou **scanf** il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdlib.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <stdio.h>`

20. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x - 1 = racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x = racine(racine(x)*racine(x));`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Le type des réels en C est :

- ☐ char  
☐ double  
☐ real  
☐ int

2. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 0  
☐ 4  
☐ 3

3. Sur unix (ou linux), la commande mkdir permet de :

- ☐ ouvrir un fichier texte  
☐ créer un répertoire  
☐ créer un fichier texte  
☐ changer de répertoire courant

4. Pour déclarer une fonction pgcd qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ int pgcd(int x, y);  
☐ int pgcd(int y, int x);  
☐ void pgcd(int x, int y);  
☐ int pgcd(int x, int x);

5. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir définie  
☐ l'avoir déclarée  
☐ avoir défini une constante symbolique de la taille de cette fonction  
☐ l'avoir déclarée et définie

6. Si x est une variable réelle (de type double) alors  $x = 3/2$  lui affecte la valeur :

- ☐ 1.5  
☐ 0.5  
☐ 0  
☐ 1

7. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE?

- ☐ (!A || B)  
☐ (A == TRUE) && (B == TRUE)  
☐ A && B  
☐ !(A || B) == (A && !B)

8. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche x  
☐ la variable x vaut  $-\frac{1}{2}$   
☐ la variable x vaut 16  
☐ le programme affiche \*\*\*\*

9. Si racine est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que x est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ x - 1 = racine(x);  
☐ x = racine(racine(x)\*racine(x));  
☐ x = racine(x \* x) - racine(x);  
☐ x = racine(2/3);

10. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression f(1) prendra la valeur :

- ☐ 1  
☐ 0  
☐ 5  
☐ 4

11. Le code suivant :

```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1 0  
☐ 0 1 2 3 4  
☐ 0 1 2 3  
☐ 4 3 2 1

12. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.

- ☐ #define N 3  
☐ #define taille = 3  
☐ #define taille = N  
☐ #define N = 3

13. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique
- ☐ un ordre quelconque

14. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 111
- ☐ 7
- ☐ 3
- ☐ 8

15. Soient deux variables entières **x** et **y** initialisées à 4 et 5 respectivement. L'affichage **x=4** et **y=5** est obtenu avec la commande :

- ☐ `printf("x=%x et y=%y\n");`
- ☐ `printf("x=%d et y=%d\n",x y);`
- ☐ `printf("x=%d et y=%d\n,x,y");`
- ☐ `printf("x=%d et y=%d\n",x,y);`

16. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 2; i = i + 1)
{
```

```
    for (j = 0; j < 3; j = j + 1)
    {
        printf("%d ", i);
    }
}
```

qu'est ce qui sera affiché ?

- ☐ 0 1 2 0 1 2
- ☐ 1 2 1 2 3
- ☐ 0 0 0 1 1 1
- ☐ 0 1 0 1 0 1 0 1

17. Dans la commande gcc, l'option **-Wall** signifie :

- ☐ qu'il faut lancer un débogueur
- ☐ qu'il faut indenter le fichier source
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'on veut changer aléatoirement de fond d'écran

18. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 3
- ☐ 20
- ☐ 6
- ☐ 16

19. Vous utilisez une boucle **while** quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un **for** dans le même programme principal
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction

20. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il risque d'afficher bonjour à la place de coucou
- ☐ il comporte une boucle infinie
- ☐ il ne compile pas
- ☐ il n'affiche rien

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int toto.n = 3;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `toto_s n, x;`
- ☐ `struct toto_s toto;`

2. Pour afficher à l'aide de `printf("%d\n",tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`

3. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(int n);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(char c);`

4. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ créer un répertoire
- ☐ changer de répertoire courant
- ☐ ouvrir un fichier texte
- ☐ créer un fichier texte

5. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir définie
- ☐ l'avoir déclarée
- ☐ l'avoir déclarée et définie
- ☐ avoir défini une constante symbolique de la taille de cette fonction

6. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=13, a=8, b=5`
- ☐ `f(a,b)=8, a=3, b=5`
- ☐ `f(3,5)=8, a=3, b=5`
- ☐ `f(a,b)=8, a=8, b=5`

7. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `char "c";`
- ☐ `char 'c';`
- ☐ `char c;`
- ☐ `int char;`

8. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
```

```
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 0
- ☐ 4
- ☐ 3

9. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `double exposant(double x, int n);`
- ☐ `exposant(double x, int n, int r);`
- ☐ `void exposant(double x^n);`
- ☐ `int exposant(double n, int x);`

10. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 16
- ☐ 0
- ☐ 4
- ☐ 8

11. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `void pgcd(int x, int y);`

12. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 0; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
...
}
```

qu'est ce qui sera affiché ?

- ☐ j = 0
- ☐ j = 5
- ☐ j = %d
- ☐ j = 4

13. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ new TP4
- ☐ yppasswd
- ☐ mkdir TP4
- ☐ kwrite TP4

14. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ main

☐ include

☐ init

☐ begin

15. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui ne sera pas détectée
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique

16. Une variable booléenne est un variable :

- ☐ NaN (not a number, qui n'est pas un nombre)
- ☐ réelle positive
- ☐ jamais nulle
- ☐ à laquelle une valeur vient d'être affectée
- ☐ qui est vraie ou fausse

17. Si pgcd est une fonction prenant en entrée deux entiers et renvoyant un entier, il est correct d'écrire :

- ☐ n = pgcd(n, 3);
- ☐ n = pgcd(int p, int q);
- ☐ int n = pgcd();
- ☐ int pgcd(2);

18. Soient deux variables entières x et y initialisées à 4 et 5 respectivement. L'affichage x=4 et y=5 est obtenu avec la commande :

- ☐ printf("x=%d et y=%d\n",x y);
- ☐ printf("x=%d et y=%d\n,x,y");
- ☐ printf("x=%x et y=%y\n");
- ☐ printf("x=%d et y=%d\n",x,y);

19. Pour déclarer une procédure afficher\_date qui prend en argument un struct date\_s et affiche le contenu du struct, on écrit :

- ☐ void afficher\_date(date\_s d);
- ☐ int afficher\_date(date\_s d);
- ☐ void afficher\_date(struct date\_s d);
- ☐ struct date\_s afficher\_date(struct date\_s d);

20. Pour l'extrait de programme suivant :

```
int produit = 1;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 0
- ☐ 4
- ☐ 16



**Barème : 1 point par réponse juste (unique) ; -0,5 point par réponse fausse. Durée : 20 minutes.**

1. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

2. Pour déclarer une variable qui sera utilisée comme variable de boucle on peut utiliser l'instruction

- ☐ `int loop n;`
- ☐ `loop i;`
- ☐ `int %d;`
- ☐ `int k;`

3. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ alphabétique
- ☐ un ordre quelconque

4. Si le code :

```
struct toto_s
{
    int n;
    double x;
};
```

précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`
- ☐ `struct toto_s toto;`
- ☐ `int toto.n = 3;`
- ☐ `toto_s struct z = {3, 0.5};`
- ☐ `toto_s n, x;`

5. Pour déclarer une fonction `pgcd` qui calcule et renvoie le plus grand diviseur commun de deux entiers positifs passés en arguments on écrit :

- ☐ `void pgcd(int x, int y);`
- ☐ `int pgcd(int y, int x);`
- ☐ `int pgcd(int x, y);`
- ☐ `int pgcd(int x, int x);`

6. Le langage C est un langage

- ☐ compilé
- ☐ lu, écrit, parlé
- ☐ composé
- ☐ interprété

7. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `struct int factorielle(int n);`
- ☐ `int factorielle(int x);`
- ☐ `int factorielle(double n);`
- ☐ `int factorielle();`

8. Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :

- ☐ `printf("Valeur de n ? %g\n", n);`
- ☐ `printf("Valeur de n ? %d\n", n);`
- ☐ un débogueur
- ☐ `scanf("%d", &n);`

9. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 0
- ☐ 7

10. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `int saisie_utilisateur();`
- ☐ `void saisie_utilisateur(char c);`

11. Les lignes

```
int i;
int x=0;
for(i=0,i<5,i=i+1)
{
    x=x+1;
}
```

- ☐ comportent une erreur qui sera détectée au cours de l'édition de lien
- ☐ ne comportent aucune erreur
- ☐ comportent une erreur qui sera détectée au cours de l'analyse syntaxique
- ☐ comportent une erreur qui ne sera pas détectée

12. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1
- ☐ 4 3 2 1 0
- ☐ 0 1 2 3 4
- ☐ 1 2 3 4

13. Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

14. Dans la commande `gcc`, l'option `-Wall` signifie :

- ☐ qu'on veut changer aléatoirement de fond d'écran
- ☐ que l'on veut voir tous les avertissements
- ☐ qu'il faut indenter le fichier source
- ☐ qu'il faut lancer un débogueur

15. Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.

- ☐ `#define N = 3`
- ☐ `#define taille = N`
- ☐ `#define taille = 3`
- ☐ `#define N 3`

16. Si cette erreur apparaît à la compilation :

**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante
- ☐ un caractère interdit en C
- ☐ une variable non déclarée
- ☐ une faute de frappe dans un appel de fonction

17. Un enregistrement permet de grouper plusieurs valeurs dans :

- ☐ ses blocs
- ☐ ses chants
- ☐ ses champs
- ☐ ses cases

18. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 6
- ☐ 16
- ☐ 20
- ☐ 3

19. Si `racine` est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que `x` est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`
- ☐ `x = racine(x * x) - racine(x);`
- ☐ `x = racine(2/3);`
- ☐ `x - 1 = racine(x);`

20. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A
- ☐ i
- ☐ cccccc
- ☐ ABCDEF

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

- Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max', que doit-on chercher dans le programme ?`
  - ☐ une directive préprocesseur `#include` manquante
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une fonction déclarée mais non définie
  - ☐ une fonction appelée avant sa déclaration
- Dans la commande gcc, l'option `-Wall` signifie :
  - ☐ qu'il faut lancer un débogueur
  - ☐ qu'on veut changer aléatoirement de fond d'écran
  - ☐ qu'il faut indenter le fichier source
  - ☐ que l'on veut voir tous les avertissements
- Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
  - ☐ `n = factorielle();`
  - ☐ `printf("%d", factorielle(n));`
  - ☐ `int factorielle(int 2);`
  - ☐ `n = factorielle(p, q);`
- Après exécution jusqu'à la ligne 15 du programme C :
 

```

10  ...
11  int main() {
12      int x = 5;
13
14      x = 3 * x + 1;
15
16      ...
17  }
```

  - ☐ le programme affiche `****`
  - ☐ la variable `x` vaut 16
  - ☐ la variable `x` vaut  $-\frac{1}{2}$
  - ☐ le programme affiche `x`

- Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
  - ☐ `char "c";`
  - ☐ `char 'c';`
  - ☐ `char c;`
  - ☐ `int char;`
- Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction
  - ☐ `char tableau[5];`
  - ☐ `int tab[] = 5;`
  - ☐ `int toto[5];`
  - ☐ `int toto[taille=5];`
  - ☐ `int[] new tableau(5);`
- Le type des réels en C est :
  - ☐ `int`
  - ☐ `double`
  - ☐ `real`
  - ☐ `char`
- Le code suivant :
 

```

int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

 affichera :
  - ☐ 0 2 4 6 8
  - ☐ 8 2
  - ☐ 8 6 4 2 0
  - ☐ 8 6 4 2
- Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :
  - ☐ alphabétique
  - ☐ dans lequel vous avez déclaré ces fonction
  - ☐ un ordre quelconque
  - ☐ dans lequel ces fonctions sont appelées dans le `main`

- Sur unix (ou linux), la commande `mkdir` permet de :
  - ☐ ouvrir un fichier texte
  - ☐ créer un fichier texte
  - ☐ changer de répertoire courant
  - ☐ créer un répertoire

- Soit la fonction `f` définie par :

```

int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

- L'ordonnancement par tourniquet permet :

- ☐ d'entretenir l'illusion que les processus tournent en parallèle
- ☐ de doubler la mémoire disponible
- ☐ de ne pas perdre de temps avec la commutation de contexte
- ☐ d'afficher des ronds colorés à l'écran

- Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studlib.h>`
- ☐ `#include <studio.h>`

14. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ ABCDEF
- ☐ i
- ☐ A
- ☐ cccccc

15. Pour l'extrait de programme suivant :

```
int i;
int j;
for(i=4;i>0;i=i-1)
{
    for(j=i;j<6;j=j+1)
    {
        printf("*");
    }
    printf(" ");
}
```

qu'est ce qui sera affiché ?

- ☐ \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*

- ☐ \*\*\*\* \*\*
- ☐ \*\* \*\*
- ☐ \*\*\*\*\*

16. Pour déclarer une fonction `saisie_utilisateur` qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ `void saisie_utilisateur(char c);`
- ☐ `saisie_utilisateur(scanf(%d));`
- ☐ `void saisie_utilisateur(int n);`
- ☐ `int saisie_utilisateur();`

17. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

18. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6 8
- ☐ 0 2 4 6
- ☐ 0 1 2 3 4 5 6

19. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 4
- ☐ 16
- ☐ 0

20. Vous utilisez une boucle `while` quand :

- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 7  
☐ 0  
☐ 5

2. Quelle étape de la compilation vient d'échouer lorsqu'on a un message comme celui-ci :

`Undefined symbols : "_printf" ou référence indéfinie vers « printf »`

- ☐ l'analyse des entrées clavier  
☐ l'édition de liens  
☐ l'analyse harmonique  
☐ l'analyse sémantique

3. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `int afficher_menu(int char);`  
☐ `int afficher_menu();`  
☐ `char afficher_menu(printf("menu"));`  
☐ `void afficher_menu();`  
☐ `double afficher_menu();`

4. Si cette erreur apparaît à la compilation : **erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une directive préprocesseur `#include` manquante  
☐ une fonction déclarée mais non définie  
☐ une fonction appelée avant sa déclaration  
☐ un désaccord entre la déclaration et la définition d'une fonction

5. Si `x` est une variable réelle (de type double) alors `x = 3/2` lui affecte la valeur :

- ☐ 1.5  
☐ 0  
☐ 0.5  
☐ 1

6. Après exécution jusqu'à la ligne 15 du programme C :

```
10    ...
11    int main() {
12        int x = 5;
13
14        x = 3 * x + 1;
15
16        ...
17    }
```

- ☐ le programme affiche `x`  
☐ la variable `x` vaut 16  
☐ le programme affiche `****`  
☐ la variable `x` vaut  $-\frac{1}{2}$

7. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 15  
☐ 10  
☐ 6  
☐ 0

8. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 111  
☐ 3  
☐ 7  
☐ 8

9. Une variable booléenne est un variable :

- ☐ NaN (not a number, qui n'est pas un nombre)  
☐ réelle positive  
☐ qui est vraie ou fausse  
☐ à laquelle une valeur vient d'être affectée  
☐ jamais nulle

10. Au début de la fonction `main()` on place le code :

```
char b = 'A';
b = b + 2;
printf("%c\n", b);
```

Alors l'affichage sera :

- ☐ A  
☐ B  
☐ C  
☐ b

11. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `printf("%d", factorielle(n));`  
☐ `n = factorielle();`  
☐ `int factorielle(int 2);`  
☐ `n = factorielle(p, q);`

12. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}
```

- ☐ il ne compile pas  
☐ il risque d'afficher bonjour à la place de coucou  
☐ il n'affiche rien  
☐ il comporte une boucle infinie

13. La virtualisation de la mémoire permet notamment de stocker des portions inactives de la mémoire de travail sur le disque dur. Mais on perd :

- ☐ certaines données de la mémoire de travail
- ☐ les fichiers du disque
- ☐ des processus
- ☐ en temps d'accès

14. Pour déclarer un tableau d'entiers de taille 5, on peut utiliser l'instruction

- ☐ `int[] new tableau(5);`
- ☐ `int tab[] = 5;`
- ☐ `int toto[5];`
- ☐ `char tableau[5];`
- ☐ `int toto[taille=5];`

15. Soit la fonction f définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression f(0) prendra la valeur :

- ☐ 3
- ☐ 0
- ☐ 4

16. Laquelle des analyses suivantes ne fait pas partie des étapes de la compilation :

- ☐ analyse harmonique
- ☐ analyse sémantique
- ☐ analyse lexicale
- ☐ analyse syntaxique

17. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ include
- ☐ init
- ☐ begin
- ☐ main

18. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C?

- ☐ `char c;`
- ☐ `char 'c';`
- ☐ `int char;`
- ☐ `char "c";`

19. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6
- ☐ 0 2 4 6
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6 8

20. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
printf("Majeur\n");
```

affichera :

- ☐ rien
- ☐ Mineur  
Majeur
- ☐ Mineur
- ☐ Majeur

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Quel est l'opérateur de différence en C :

- ☐  $\neq$
- ☐  $\lt \gt$
- ☐  $!$
- ☐  $!=$

2. Pour l'extrait de programme suivant :

```
int produit = 0;
int serie[4] = {2, 2, 2, 2};
for (i = 0; i < 4; i = i + 1)
{
    produit = produit * serie[i];
}
printf("produit = %d", produit);
```

La valeur affichée est :

- ☐ 8
- ☐ 16
- ☐ 0
- ☐ 4

3. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 8
- ☐ 7
- ☐ 111
- ☐ 3

4. Pour afficher à l'aide de `printf("%d\n", tab[i]);` le contenu d'un tableau de 5 entiers initialisé au préalable, on utilise plutôt :

- ☐ `for(i=0;i<5;i=i+1)`
- ☐ `for(i=0;i<=5;i=i+1)`
- ☐ `for(i=1;i<=5;i=i+1)`
- ☐ `for(i=1;i<5;i=i+1)`

5. On considère deux variables booléennes A et B initialisées à TRUE et FALSE respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur TRUE ?

- ☐ `(!A || B)`
- ☐ `A && B`
- ☐ `!(A || B) == (A && !B)`
- ☐ `(A == TRUE) && (B == TRUE)`

6. Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ Majeur
- ☐ Mineur
- ☐ Mineur Majeur
- ☐ rien

7. Le langage C est un langage

- ☐ compilé
- ☐ interprété
- ☐ lu, écrit, parlé
- ☐ composé

8. Vous utilisez une boucle `while` quand :

- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous avez déjà fait un `for` dans le même programme principal

9. Un bit est :

- ☐ la longueur d'un mot mémoire
- ☐ l'instruction qui met fin à un programme
- ☐ un chiffre binaire (0 ou 1)
- ☐ un battement d'horloge processeur

10. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studio.h>`
- ☐ `#include <studlib.h>`

11. Après exécution jusqu'à la ligne 15 du programme C :

```
10 int main() {
11     int x = 5;
12     int y;
13
14     y = x;
15
16     ...
17 }
```

- ☐ la variable y vaut 5
- ☐ le programme affiche "Faux"
- ☐ la variable x vaut 0
- ☐ la variable x vaut 5 et la variable y vaut 0

12. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que n est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`
- ☐ `int carre(2);`
- ☐ `int n = carre();`

13. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 4
- ☐ 101
- ☐ 3
- ☐ 5

14. Un enregistrement permet de grouper plusieurs valeurs dans :
- ☐ ses champs
  - ☐ ses chants
  - ☐ ses cases
  - ☐ ses blocs
15. Afin de représenter la taille d'un tableau, définir une constante symbolique N valant 3.
- ☐ `#define taille = 3`
  - ☐ `#define N = 3`
  - ☐ `#define N 3`
  - ☐ `#define taille = N`
16. Pour l'extrait de programme suivant :
- ```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```
- La valeur de somme affichée est :
- ☐ 3

- ☐ 6
  - ☐ 20
  - ☐ 16
17. Avant de faire appel à une fonction il est nécessaire de :
- ☐ l'avoir définie
  - ☐ l'avoir déclarée et définie
  - ☐ l'avoir déclarée
  - ☐ avoir défini une constante symbolique de la taille de cette fonction
18. Si a et b sont deux variables de type :
- ```
struct toto_s
{
    int n;
    double x;
};
```
- Alors pour tester l'égalité de a et de b on utilise la condition :
- ☐ `a{n, x} == b{n, x}`
  - ☐ `a == b`
  - ☐ `(a.n == b.n) && (a.x == b.x)`
  - ☐ `a = b`

19. Si cette erreur apparaît à la compilation :  
**erreur: conflicting types for 'max'** , que doit-on chercher dans le programme ?
- ☐ une fonction appelée avant sa déclaration
  - ☐ une fonction déclarée mais non définie
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une directive préprocesseur **#include** manquante
20. Soit la fonction g définie par :
- ```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```
- Alors l'expression g(0) prendra la valeur :
- ☐ 5
  - ☐ 7
  - ☐ 0



**Barème : 1 points par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `n = factorielle();`  
☐ `int factorielle(int 2);`  
☐ `printf("%d", factorielle(n));`  
☐ `n = factorielle(p, q);`

2. Un fichier source est :

- ☐ un document qui doit être protégé  
☐ un document illisible pour les humains  
☐ un fichier que l'ont doit citer dans les documents produits sur l'ordinateur  
☐ un fichier texte qui sera traduit en instructions processeur  
☐ un document de référence du système

3. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction `f` ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ `f(a,b)=8, a=8, b=5`  
☐ `f(a,b)=13, a=8, b=5`  
☐ `f(a,b)=8, a=3, b=5`  
☐ `f(3,5)=8, a=3, b=5`

4. Pour déclarer une fonction `exposant` qui prend en argument un réel  $x$  et un entier positif  $n$  et renvoie la valeur de  $x^n$  on écrit :

- ☐ `exposant(double x, int n, int r);`  
☐ `double exposant(double x, int n);`  
☐ `int exposant(double n, int x);`  
☐ `void exposant(double x^n);`

5. Le code suivant :

```
int i;
for (i = 4; i >= 0; i = i - 1)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 4 3 2 1  
☐ 0 1 2 3 4  
☐ 1 2 3 4  
☐ 4 3 2 1 0

6. Sous unix (ou linux), pour créer un répertoire TP4 dans le répertoire courant on peut utiliser la commande :

- ☐ `kwrite TP4`  
☐ `yppasswd`  
☐ `mkdir TP4`  
☐ `new TP4`

7. Pour déclarer une fonction `factorielle` qui prend en argument un entier et renvoie sa factorielle on écrit :

- ☐ `int factorielle(int x);`  
☐ `int factorielle(double n);`  
☐ `struct int factorielle(int n);`  
☐ `int factorielle();`

8. Au début de la fonction `main()` on place le code :

```
char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");
```

Alors l'affichage sera :

- ☐ A  
☐ i  
☐ cccccc  
☐ ABCDEF

9. L'écriture 101 en binaire correspond au nombre naturel :

- ☐ 5  
☐ 4  
☐ 3  
☐ 101

10. Pour l'extrait de programme suivant :

```
int somme = 0;
for (i = 0; i < 5; i = i + 1)
{
    somme = somme + i;
    i = i + 1; /* attention ! */
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 15  
☐ 10  
☐ 0  
☐ 6

11. Avant de faire appel à une fonction il est nécessaire de :

- ☐ l'avoir déclarée  
☐ avoir défini une constante symbolique de la taille de cette fonction  
☐ l'avoir définie  
☐ l'avoir déclarée et définie

12. Vous utilisez une boucle **while** quand :

- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
- ☐ vous n'avez pas déclaré de fonction
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous avez déjà fait un **for** dans le même programme principal

13. Sur unix (ou linux), la commande **mkdir** permet de :

- ☐ créer un fichier texte
- ☐ ouvrir un fichier texte
- ☐ créer un répertoire
- ☐ changer de répertoire courant

14. Pour l'extrait de programme suivant :

```
int somme = 0;
int serie[4] = {2, 4, 10, 4};
for (i = 0; i < 4; i = i + 1)
{
    somme = somme + serie[i];
}
printf("somme = %d",somme);
```

La valeur de somme affichée est :

- ☐ 20
- ☐ 16
- ☐ 3
- ☐ 6

15. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Arriver à l'heure en cours

16. Si cette erreur apparaît à la compilation :

**Undefined symbols : "\_printf" ou référence indéfinie vers « printf »** que doit-on chercher dans le programme ?

- ☐ un caractère interdit en C
- ☐ une faute de frappe dans un appel de fonction
- ☐ une variable non déclarée
- ☐ une directive préprocesseur **#include** manquante

17. Pour compiler un programme **prog.c**, on utilise la ligne de commande :

- ☐ **gcc -Wall prog.c -o prog.exe**
- ☐ **gcc -Wall prog.exe -o prog.c**
- ☐ **gcc prog.c -o -Wall prog.exe**
- ☐ **gcc prog.exe -Wall -o prog.c**

18. Si **x** est une variable réelle (de type double) alors **x = 3/2** lui affecte la valeur :

- ☐ 1.5
- ☐ 0
- ☐ 0.5
- ☐ 1

19. Après exécution jusqu'à la ligne 15 du programme C :

```
10  int main() {
11      int x = 5;
12      int y;
13
14      y = x;
15
16      ...
17  }
```

- ☐ la variable **x** vaut 5 et la variable **y** vaut 0
- ☐ la variable **x** vaut 0
- ☐ le programme affiche "Faux"
- ☐ la variable **y** vaut 5

20. Pour déclarer une fonction **saisie\_utilisateur** qui demande à l'utilisateur d'entrer un entier au clavier et renvoie cet entier on écrit :

- ☐ **saisie\_utilisateur scanf(%d);**
- ☐ **void saisie\_utilisateur(int n);**
- ☐ **void saisie\_utilisateur(char c);**
- ☐ **int saisie\_utilisateur();**

**Barème : 1 point par réponse juste (unique) ; −0,5 points par réponse fausse. Durée : 20 minutes.**

- Si cet avertissement apparaît à la compilation :  
`warning: implicit declaration of function 'max'`  
, que doit-on chercher dans le programme ?
  - ☐ une fonction déclarée mais non définie
  - ☐ une fonction appelée avant sa déclaration
  - ☐ un désaccord entre la déclaration et la définition d'une fonction
  - ☐ une directive préprocesseur `#include` manquante
- Le type des réels en C est :
  - ☐ `double`
  - ☐ `char`
  - ☐ `real`
  - ☐ `int`
- Vous utilisez une boucle `while` quand :
  - ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance
  - ☐ vous n'avez pas déclaré de fonction
  - ☐ vous avez déjà fait un `for` dans le même programme principal
  - ☐ l'incrément de la variable de boucle n'est pas 1
- Le code suivant :

```
int i;
for (i = 8; i > 0; i = i - 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :
  - ☐ 8 2
  - ☐ 0 2 4 6 8
  - ☐ 8 6 4 2
  - ☐ 8 6 4 2 0

- On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `(a<=n) && (n<=b)`
  - ☐ `a<=n<=b`
  - ☐ `(n<=a) && (n<=b)`
  - ☐ `(a<n) || (n>b)`
- Un enregistrement permet de grouper plusieurs valeurs dans :
    - ☐ ses champs
    - ☐ ses chants
    - ☐ ses blocs
    - ☐ ses cases
  - Afin de représenter la taille d'un tableau, définir une constante symbolique `N` valant 3.
    - ☐ `#define taille = N`
    - ☐ `#define taille = 3`
    - ☐ `#define N = 3`
    - ☐ `#define N 3`
  - Soit la fonction `f` définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return 3;
    }
    return 4;
}
```

Alors l'expression `f(0)` prendra la valeur :

- ☐ 4
- ☐ 3
- ☐ 0

- Le code suivant :

```
int somme = 0;
int i;
for (i = 1; i < 4; i = i + 1)
{
    somme = somme + i;
}
printf("%d", somme);
```

affichera :

- ☐ 6
- ☐ 0
- ☐ 42
- ☐ 1

- Si `n` est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
  - ☐ un débogueur
  - ☐ `printf("Valeur de n ? %d\n", n);`
  - ☐ `printf("Valeur de n ? %g\n", n);`
  - ☐ `scanf("%d", &n);`
- On considère deux variables booléennes `A` et `B` initialisées à `TRUE` et `FALSE` respectivement. Parmi les expressions booléennes suivantes, laquelle a pour valeur `TRUE` ?
  - ☐ `(A == TRUE) && (B == TRUE)`
  - ☐ `A && B`
  - ☐ `(!A || B)`
  - ☐ `!(A || B) == (A && !B)`
- Le code suivant :

```
int age = 15;
if (age < 18)
{
    printf("Mineur\n");
}
```

```

}
else
{
    printf("Majeur\n");
}

```

affichera :

- ☐ Mineur
- ☐ Majeur
- ☐ rien
- ☐ Mineur
- ☐ Majeur

13. Le code suivant :

```

int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}

```

affichera :

- ☐ Mineur
- ☐ rien
- ☐ Mineur
- ☐ Majeur
- ☐ Majeur

14. Au début de la fonction `main()` on place le code :

```

char i;
for (i = 'A'; i <= 'F'; i = i + 1)
{
    printf("%c", i);
}
printf("\n");

```

Alors l'affichage sera :

- ☐ A
- ☐ ABCDEF
- ☐ i
- ☐ cccccc

15. Soit la fonction `g` définie par :

```

int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}

```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

16. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire
- ☐ Écrire des données sur le disque dur

☐ transporter les processus du tourniquet au processeur

☐ Arriver à l'heure en cours

17. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :

- ☐ `double afficher_menu();`
- ☐ `char afficher_menu(printf("menu"));`
- ☐ `int afficher_menu(int char);`
- ☐ `void afficher_menu();`
- ☐ `int afficher_menu();`

18. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :

- ☐ `int factorielle(int 2);`
- ☐ `n = factorielle(p, q);`
- ☐ `n = factorielle();`
- ☐ `printf("%d", factorielle(n));`

19. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `main`
- ☐ `include`
- ☐ `begin`
- ☐ `init`

20. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <studio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <stdio.h>`
- ☐ `#include <studlib.h>`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Une *segmentation fault* est une erreur qui survient lorsque :

- ☐ la division du programme en zones homogènes échoue
- ☐ le programme tente d'afficher des caractères sur une ligne qui va au delà de la largeur de la fenêtre du terminal
- ☐ le programme tente d'accéder à une partie de la mémoire qui ne lui est pas réservée
- ☐ le programme source a été enregistré sur le disque dur en plusieurs morceaux et l'un d'entre eux ne peut pas être chargé par le compilateur

2. Soit la fonction `g` définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression `g(0)` prendra la valeur :

- ☐ 5
- ☐ 7
- ☐ 0

3. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel vous avez déclaré ces fonction
- ☐ alphabétique
- ☐ un ordre quelconque
- ☐ dans lequel ces fonctions sont appelées dans le main

4. Vous utilisez une boucle `while` quand :

- ☐ vous avez déjà fait un `for` dans le même programme principal
- ☐ l'incrément de la variable de boucle n'est pas 1
- ☐ vous n'avez pas déclaré de fonction
- ☐ vous ne connaissez pas le nombre d'itérations de la boucle à l'avance

5. Si cette erreur apparaît à la compilation :

**erreur: conflicting types for 'max'**, que doit-on chercher dans le programme ?

- ☐ une fonction appelée avant sa déclaration
- ☐ une directive préprocesseur `#include` manquante
- ☐ un désaccord entre la déclaration et la définition d'une fonction
- ☐ une fonction déclarée mais non définie

6. Une de ces manière de composer les blocs de programmes ne fait pas partie des opérations de la programmation structurée :

- ☐ sélectionner entre deux blocs à l'aide d'une condition
- ☐ mettre les blocs en séquence les uns à la suite des autres
- ☐ retourner un bloc
- ☐ répéter un bloc tant qu'une condition est vérifiée

7. Un registre du processeur est :

- ☐ un composant qui contient la liste des fichiers du système
- ☐ une unité de calcul spécialisée de l'ordinateur
- ☐ une gamme de fréquence de fonctionnement du processeur
- ☐ une case mémoire interne au processeur qui sera manipulée directement lors des calculs

8. Le bus système sert à :

- ☐ Écrire des données sur le disque dur
- ☐ Arriver à l'heure en cours
- ☐ transporter les processus du tourniquet au processeur
- ☐ Transférer des données et intructions entre processeur et mémoire

9. Si `carre` est une fonction prenant en entrée un entier et renvoyant le carré de cet entier, et que `n` est une variable entière définie et initialisée, il est correct d'écrire :

- ☐ `int n = carre();`
- ☐ `int carre(2);`
- ☐ `n = carre(n);`
- ☐ `n = carre(int n);`

10. Le code suivant :

```
int i;
for (i = 0; i < 7; i = i + 2)
{
    printf("%d ", i);
}
printf("\n");
```

affichera :

- ☐ 0 1 2 3 4 5 6
- ☐ 0 1 2 3 4 5 6 7
- ☐ 0 2 4 6 8
- ☐ 0 2 4 6

11. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

12. Sur unix (ou linux), la commande `mkdir` permet de :

- ☐ ouvrir un fichier texte
- ☐ créer un fichier texte
- ☐ changer de répertoire courant
- ☐ créer un répertoire

13. Quels calculs peut-on programmer en programmation structurée ?
- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
  - ☐ certains programmes sont de vrais plats de spaghetti
  - ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
  - ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine
14. Pour déclarer une procédure `afficher_menu` sans argument et qui ne renvoie rien on utilise :
- ☐ `int afficher_menu(int char);`
  - ☐ `double afficher_menu();`
  - ☐ `char afficher_menu(sprintf("menu"));`
  - ☐ `void afficher_menu();`
  - ☐ `int afficher_menu();`
15. Si  $n$  est une variable entière pour demander sa valeur à l'utilisateur, on utilise plutôt :
- ☐ `printf("Valeur de n ? %d\n", n);`
  - ☐ `scanf("%d", &n);`
  - ☐ un débogueur
  - ☐ `printf("Valeur de n ? %g\n", n);`

16. Laquelle de ces écritures correspond à la déclaration d'une variable de type caractère en langage C ?
- ☐ `char "c";`
  - ☐ `char 'c';`
  - ☐ `int char;`
  - ☐ `char c;`
17. Le code suivant :
- ```
int i;
for (i = 0; i < 5; i = i + 1)
{
    printf("%d ", i);
}
printf("\n");
```
- affichera :
- ☐ 4 3 2 1 0
  - ☐ 0 1 2 3 4
  - ☐ 0 1 2 3
  - ☐ 4 3 2 1
18. Si le code :
- ```
struct toto_s
{
    int n;
    double x;
};
```
- précède la fonction `main()`, alors on peut écrire en début de `main()` :

- ☐ `int struct toto_s = {3, -1e10};`
  - ☐ `toto_s struct z = {3, 0.5};`
  - ☐ `struct toto_s toto;`
  - ☐ `toto_s n, x;`
  - ☐ `int toto.n = 3;`
19. Si `factorielle` est une fonction prenant en entrée un entier et renvoyant un entier, il est correct d'écrire :
- ☐ `n = factorielle();`
  - ☐ `n = factorielle(p, q);`
  - ☐ `int factorielle(int 2);`
  - ☐ `printf("%d", factorielle(n));`
20. Si  $a$  et  $b$  sont deux variables de type :
- ```
struct toto_s
{
    int n;
    double x;
};
```
- Alors pour tester l'égalité de  $a$  et de  $b$  on utilise la condition :
- ☐ `a == b`
  - ☐ `a{n, x} == b{n, x}`
  - ☐ `a = b`
  - ☐ `(a.n == b.n) && (a.x == b.x)`

**Barème : 1 point par réponse juste (unique) ; -0,5 points par réponse fausse. Durée : 20 minutes.**

1. Si **racine** est une fonction prenant en entrée un réel et renvoyant la racine carrée de cet réel, et que **x** est une variable réelle définie et initialisée, il est incorrect d'écrire :

- ☐ `x = racine(racine(x)*racine(x));`  
☐ `x - 1 = racine(x);`  
☐ `x = racine(2/3);`  
☐ `x = racine(x * x) - racine(x);`

2. Le bus système sert à :

- ☐ Transférer des données et intructions entre processeur et mémoire  
☐ Arriver à l'heure en cours  
☐ Écrire des données sur le disque dur  
☐ transporter les processus du tourniquet au processeur

3. Un programme en langage C doit comporter une et une seule définition de la fonction :

- ☐ `main`  
☐ `init`  
☐ `include`  
☐ `begin`

4. Soit un programme contenant les lignes suivantes :

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i = i + 1)
{
    for (j = 0; j < 5; j = j + 1)
    {
        ...
    }
}
printf("j = %d\n", j);
```

qu'est ce qui sera affiché par ce printf?

- ☐ `j = 4`  
☐ `j = %d`  
☐ `j = 5`  
☐ `j = 0`

5. Soit la fonction **f** définie par :

```
int f(int a)
{
    printf("a = \n", %d);
    if (a > 0)
    {
        return f(a - 1) + 1;
    }
    return 4;
}
```

Alors l'expression **f(1)** prendra la valeur :

- ☐ 0  
☐ 5  
☐ 1  
☐ 4

6. Dans la commande gcc, l'option **-Wall** signifie :

- ☐ qu'il faut lancer un débogueur  
☐ qu'il faut indenter le fichier source  
☐ que l'on veut voir tous les avertissements  
☐ qu'on veut changer aléatoirement de fond d'écran

7. Quel est le problème d'un programme comportant les lignes suivantes ?

```
while (1)
{
    printf("coucou\n");
}

☐ il ne compile pas
☐ il n'affiche rien
☐ il comporte une boucle infinie
☐ il risque d'afficher bonjour à la place de coucou
```

8. Soit la fonction **g** définie par :

```
int g(int a)
{
    printf("a = \n", %d);
    if (1 > 0)
    {
        return 5;
    }
    return 7;
}
```

Alors l'expression **g(0)** prendra la valeur :

- ☐ 0  
☐ 7  
☐ 5

9. Le code suivant :

```
int age = 20;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien  
☐ Majeur  
☐ Mineur  
Majeur  
☐ Mineur

10. Quel est l'opérateur de différence en C :

- ☐ `≠`  
☐ `!=`  
☐ `<>`  
☐ `!`

11. Soit le programme principal suivant :

```
int main()
{
    int a = 3;
    int b = 5;
    printf("f(a,b)=%d, a=%d, b=%d\n",f(a,b),a,b);
    return EXIT_SUCCESS;
}
```

appelant la fonction f ainsi définie :

```
int f(int a, int b)
{
    a = a + b;
    return a;
}
```

L'affichage dans le main est le suivant :

- ☐ f(a,b)=13, a=8, b=5
- ☐ f(a,b)=8, a=8, b=5
- ☐ f(a,b)=8, a=3, b=5
- ☐ f(3,5)=8, a=3, b=5

12. Lorsqu'un programme utilise `printf` ou `scanf` il faut qu'il contienne l'instruction préprocesseur :

- ☐ `#include <studio.h>`
- ☐ `#include <stdio.h>`
- ☐ `#appart <stdlib.h>`
- ☐ `#include <studlib.h>`

13. On souhaite faire une boucle de contrôle de saisie : tant que l'entier `n` n'appartient pas à l'intervalle  $[a..b]$ , on recommence la saisie de `n`. Soit le programme suivant :

```
int a = 0;
int b = 20;
int n;
scanf("%d", &n);
while(cond)
{
    scanf("%d", &n);
}
```

Quelle est la condition `cond` :

- ☐ `a<=n<=b`
- ☐ `(a<=n) && (n<=b)`
- ☐ `(n<=a) && (n<=b)`
- ☐ `(a<n) || (n>b)`

14. Si `a` et `b` sont deux variables de type :

```
struct toto_s
{
    int n;
    double x;
};
```

Alors pour tester l'égalité de `a` et de `b` on utilise la condition :

- ☐ `a == b`
- ☐ `a = b`
- ☐ `(a.n == b.n) && (a.x == b.x)`
- ☐ `a{n, x} == b{n, x}`

15. Quels calculs peut-on programmer en programmation structurée ?

- ☐ il y a des calculs programmables en programmation structurée qui ne sont pas programmables en langage machine
- ☐ certains programmes sont de vrais plats de spaghetti
- ☐ il y a des calculs programmables en langage machine et qui ne sont pas programmables en programmation structurée
- ☐ en programmation structurée on peut programmer tous les calculs programmables en langage machine

16. Pour déclarer une procédure `afficher_date` qui prend en argument un `struct date_s` et affiche le contenu du struct, on écrit :

- ☐ `void afficher_date(date_s d);`
- ☐ `void afficher_date(struct date_s d);`
- ☐ `int afficher_date(date_s d);`
- ☐ `struct date_s afficher_date(struct date_s d);`

17. Avant de faire appel à une fonction il est nécessaire de :

- ☐ avoir défini une constante symbolique de la taille de cette fonction
- ☐ l'avoir définie
- ☐ l'avoir déclarée
- ☐ l'avoir déclarée et définie

18. Vous avez déclaré préalablement un ensemble de fonctions utilisées par votre programme principal. L'ordre dans lequel vous devez maintenant définir ces fonctions est l'ordre :

- ☐ dans lequel ces fonctions sont appelées dans le main
- ☐ dans lequel vous avez déclaré ces fonction
- ☐ un ordre quelconque
- ☐ alphabétique

19. Le code suivant :

```
int age = 18;
if (age < 18)
{
    printf("Mineur\n");
}
else
{
    printf("Majeur\n");
}
```

affichera :

- ☐ rien
- ☐ Mineur
- ☐ Majeur
- ☐ Mineur Majeur

20. L'écriture 111 en binaire correspond au nombre naturel :

- ☐ 3
- ☐ 8
- ☐ 7
- ☐ 111