
Éléments d'informatique : partiel de fin de semestre

Durée : 3 heures.

Documents autorisés : Aucun.

Recommandations : Un barème vous est donné à titre indicatif, afin de vous permettre de gérer votre temps. Ne dépassez pas les temps indiqués.


1 Étude de programmes et questions de cours

1.1 Somme des éléments d'un tableau (2,5 points)

Nous voulons écrire un programme qui calcule la somme des éléments d'un tableau d'entiers. Une partie du programme est déjà écrite, et Pippo pense qu'il ne reste plus qu'à écrire la partie qui calcule effectivement la somme. Voici son programme :

```
1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf, scanf */
3
4  /* déclaration constantes et types utilisateurs */
5  #define TAILLE 3
6
7  /* Fonction principale */
8  int main()
9  {
10     int t[TAILLE] = {12,10,15}; /* tableau à sommer */
11
12
13
14     /* calcul de la somme (À FAIRE) */
15
16
17
18
19
20     /* affichage du résultat */
21     printf("La somme est %d\n", somme);
22
23     /* valeur fonction */
24     return EXIT_SUCCESS;
25 }
```

Question A. Avant d'aller plus loin Pippo veut tester son programme, mais la compilation échoue. Expliquer pourquoi, quelle étape de la compilation échoue précisément et ce qu'il manque pour que la compilation réussisse.

 1 pt
9 min


Correction. Il manque la déclaration (et l'initialisation à zéro) de la variable `somme`. La compilation échoue à l'analyse sémantique.

Barème.

0.5 pt pour « déclaration de la variable `somme` »

0.5 pt pour « analyse sémantique » ou *0.25 pt* si « avant la génération du code objet » et/ou « avant l'édition de lien ».

Question B. Compléter le programme pour qu'il calcule effectivement la somme des éléments du tableau (pour une taille de tableau arbitraire).

 **1.5 pt**
13 min

Un exemple de sortie du programme pour le tableau donné dans le code est :

La somme est 37

Correction.

```
11     int somme = 0; /* accumulateur pour la somme */
12     int i; /* var de boucle */
13
14     for (i = 0; i < TAILLE; i = i + 1) /* pour chaque case */
15     {
16         somme = somme + t[i]; /* ajout de la case a la somme */
17     }
18
```

Barème. 1.5 pt pour le code juste (compilation correction quel que soit `TAILLE`). Pas de points en moins ou en plus pour la déclaration de la variable de boucle (redite sous une autre forme de la première question) ou pour l'initialisation de `somme` à zéro. Sinon, maximum 1 pt :

zéro si pas de boucle mettre zéro à la question

0.5 pt une (et une seule) boucle (*for*, *while* accepté)


+0.5 pt la boucle parcourt effectivement un tableau de taille `TAILLE`

1.2 Une erreur classique (1 points)

Pippo a écrit un programme C. Celui-ci compile, mais une erreur survient à l'exécution, qu'il ne comprend pas.

```
$ gcc puissance.c -o puissance.exe
$ puissance.exe
Entrer un nombre reel : 2.3
Entrer son exposant (entier positif) : 2
Segmentation fault
```

Question C. Expliquer brièvement ce que signifie ce message d'erreur (dernière ligne).

 **1 pt**
9 min

Correction. Le message d'erreur `Segmentation fault` signifie que le programme a tenté de lire ou d'écrire dans un espace mémoire qui ne lui était pas réservé, ce que le système a détecté et refusé, provoquant la terminaison prématurée du programme et l'affichage du message d'erreur.

Barème. Maximum 1 pt.


1 pt *il faut que la réponse parle de mémoire et soit de (non) réservation/allocation au processus/programme, soit d'intervention du système d'exploitation.*

0.5 pt *s'il est juste fait mention de la mémoire (sans réservation etc.).*

+0.25 pt *si ça parle d'une erreur dans l'écriture d'un scanf.*

2 Trace d'un programme avec fonctions (4 points)

Question D. Simulez l'exécution du programme figure 1 page 3, en réalisant sa **trace**, comme cela a été vu en TD et en cours.

 4 pt
36 min

```
1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf, scanf */
3
4  /* declarations constantes et types utilisateurs */
5
6  /* declarations de fonctions utilisateurs */
7  int foo(int a, int b);
8  int bar(int n);
9
10 int main()
11 {
12     int x = 2;
13     int y = 6;
14     int res;
15     res = foo(x, y);
16     printf("foo(%d, %d) = %d\n", x, y, res);
17     res = bar(x);
18     printf("bar(%d) = %d\n", x, res);
19
20     return EXIT_SUCCESS;
21 }
22
23 /* definitions de fonctions utilisateurs */
24 int foo(int a, int b)
25 {
26     if (a < b)
27     {
28         return a;
29     }
30     return b;
31 }
32
33 int bar(int n)
34 {
35     if (n > 1)
36     {
37         return n * bar(n - 1);
38     }
39     return 1;
40 }
```

FIGURE 1 – Programme pour la trace

main()							
ligne	x	y	res	Affichage			
ini.	2	6	?				
15					foo(2, 6)		
	ligne	a	b				
	ini.	2	6				
	28	renvoie 2					
15			2				
16				foo(2, 6) = 2\n			
17					bar(2)		
	ligne	n					
	ini.	2					
	37				bar(1)		
		ligne	n				
		ini.	1				
		39	renvoie 1				
	37	renvoie 2					
17			2				
18				bar(2) = 2\n			
20	renvoie EXIT_SUCCESS						

TABLE 1 – Trace du programme de l'exercice 2.

Correction. Table 1 page 4.

Barème. Maximum 4 pt. Si des erreurs, maximum 3,75 pt.

- (a) +1 pt deux premières lignes de la trace du main sont correctes (identification des variables et leurs initialisations).
- * −0,5 pt par variable manquante ou en trop
 - * −0,5 pt par initialisation manquante ou en trop.
- (b) +1,25 pt **Pour l'appel à foo :**
- * +0,25 pt pour foo(2, 6) ligne 15
 - * +0,25 pt pour les deux colonnes pour les paramètres formels a et b
 - * +0,25 pt pour l'initialisation correcte des paramètres formels a et b
 - * +0,25 pt pour le retour d'un entier (même si valeur fausse)
 - * +0,25 pt ligne 15 affectation en accord avec la valeur retournée.
- (c) +1,25 pt **Pour l'appel récursif à bar :**
- * +0,25 pt pour bar(2) ligne 17
 - * +0,5 pt pour le déclenchement du second appel ligne 37, compter uniquement +0,25 pt si il y a plus de deux appels
 - * +0,25 pt une colonne pour le paramètre formel n bien initialisé à 2 ou à 1 dans le second appel (l'un des deux suffit)
 - * +0,25 pt pour le retour de la valeur et l'affectation en accord avec la valeur retournée ligne 17
- (d) +0,25 pt pour au moins l'un des deux affichages (ligne 16 et ligne 18) avec des valeurs cohérentes avec le contenu des colonnes du main.

3 For ou while ? (4,5 points)

Il est demandé de résoudre les questions suivantes sans définir de fonctions utilisateurs, directement dans le `main`, et en faisant le meilleur choix entre `for` et `while`.

Question E. Écrire un programme qui étant donné deux variables entières `largeur` et `hauteur`, initialisées à des valeurs de votre choix, affiche un rectangle d'étoiles de dimension `largeur` par `hauteur`. Dans l'exemple d'affichage suivant `largeur` vaut 6 et `hauteur` vaut 4.

1,5 pt
13 min

```
*****
*****
*****
*****
```

Correction.

```
1  /* Declaration de fonctionnalites supplementaires */
2  #include <stdlib.h> /* pour EXIT_SUCCESS */
3  #include <stdio.h> /* pour printf() et scanf() */
4
5  /* Fonction principale */
6  int main()
7  {
8      int largeur = 6;
9      int hauteur = 4;
10     int i; /* var de boucle */
11     int j; /* var de boucle */
12     for (i = 0; i < hauteur; i = i + 1) /* pour chaque ligne */
13     {
14         /* afficher une ligne de largeur etoiles */
15         for (j = 0; j < largeur; j = j + 1) /* repeter largeur fois */
16         {
17             printf("*");
18         }
19         printf("\n"); /* fin de la ligne */
20     }
21
22     return EXIT_SUCCESS;
23 }
24
```

Barème.

zéro Si pas de boucle

1,5 pt Si le programme fonctionne correctement, avec deux variables (pas des constantes symboliques), `largeur` et `hauteur`, pour n'importe quelles valeurs positives de ces variables, même s'il y a des trucs inutiles autour (saisie par l'utilisateur, constantes symboliques, etc.).

0,5 pt Si le programme contient deux boucles imbriquées
+**0,25 pt** si ce sont des `for`.

+**0,25 pt** si l'une des boucles du programme est répétée `largeur` ou `hauteur` fois avec une variable de boucle (ou un compteur si c'est un `while`) correctement déclaré,

+**0,25 pt** si `largeur` et `hauteur` sont déclarées et initialisées comme variables entières.

Question F. Expliquer comment modifier le programme précédent pour qu'il affiche le contour du rectangle avec des étoiles et l'intérieur avec des espaces.

1 pt
9 min

```
*****
*      *
*      *
*****
```

Correction. On remplace la ligne 17 du programme précédent par les lignes suivantes.

```
17         if ( (0 == i)
18             || (0 == j)
19             || (hauteur - 1 == i)
20             || (largeur - 1 == j))
21         {
22             printf("*");
23         }
24         else
25         {
26             printf(" ");
27         }
28
```

Ainsi si *i* ou *j* sont sur un bord du rectangle c'est une étoile qui sera affichée, un espace blanc autrement.

Barème. Il y a deux types de solutions : soit faire plus de boucles successives (cas particuliers), soit placer une ou plusieurs conditionnel dans le code interne à la (double) boucle. Toutes les deux sont admises et valent au maximum un point.

Pour la première solution compter 0,25 pt par bord correctement dessiné : (première ligne et dernière ligne comme boucles à part, lignes du milieu : étoile initial à part, ligne de blancs, étoile finale à part).

Pour la seconde solution, compter 0,5 pt si la solution repose sur un ou plusieurs if else (enlever 0,25 pt si seul le cas if est traité, sans tenir compte des espaces blancs) et compter +0,25 pt si il y a seulement une des erreurs suivantes : utilisation d'un et logique au lieu du ou, ou bien deux bords faux au maximum.

Question G. Écrire un programme qui, étant donné un tableau d'entier initialisé à des valeurs de votre choix, demande à l'utilisateur de saisir un entier puis si l'entier saisi est dans le tableau affiche son indice et sinon affiche à l'utilisateur **entier absent** du tableau.

2 pt
18 min

Correction.

```
1  /* Declaration de fonctionnalites supplementaires */
2  #include <stdlib.h> /* pour EXIT_SUCCESS */
3  #include <stdio.h> /* pour printf() et scanf() */
4
5  /* Declarations de constantes */
6  #define TAILLE 5
7  #define TRUE 1
8  #define FALSE 0
```

```

9  /* Fonction principale */
10 int main()
11 {
12     int tab[TAILLE] = {1, 2, 4, 2, 5};
13     int x; /* valeur cherchée */
14     int trouve = FALSE;
15     int i = 0; /* var de boucle */
16
17     /* saisie utilisateur */
18     printf("entier recherché ? ");
19     scanf("%d", &x);
20
21     /* recherche dans le tableau */
22     while ((!trouve) && (i < TAILLE))
23     /* tant qu'on n'a pas trouvé et qu'il reste des éléments à
24      * parcourir */
25     {
26         if (x == tab[i])
27         {
28             trouve = TRUE;
29         }
30         else
31         {
32             i = i + 1;
33         }
34     }
35
36     /* affichage du résultat */
37     if (trouve)
38     {
39         printf("entier %d trouvé à l'indice %d\n", x, i);
40     }
41     else
42     {
43         printf("entier absent du tableau\n");
44     }
45
46     return EXIT_SUCCESS;
47 }

```

Barème. On propose un corrigé avec séparation du traitement et de l'affichage et une variable booléenne pour réguler la poursuite de la recherche, qui du coup nécessite des variables booléennes. Une solution plus compacte est admise si elle fonctionne. Dans ce cas, si elle repose effectivement sur un parcours partiel du tableau (normalement à base de `while`), on compte 2 pt, si le programme parcourt tout le tableau (solution de type `for`) on compte 1,5 pt.

Si la solution ne fonctionne pas, on travaille avec le barème suivant, en **limitant** le total des points à 1,75 pt.

Déclaration du tableau 0,5 pt offerts (il suffit de s'inspirer de l'énoncé du premier exercice)

0,25 pt déclaration et initialisation correcte du tableau

+0,25 pt si utilise une constante symbolique préalablement déclarée pour la taille du tableau

Saisie 0, 5 pt

0,25 pt pour une (tentative de) saisie : `printf` suivi de `scanf` avec référence à une variable de type entier dans le `scanf`, préalablement déclarée

0,25 pt si le `scanf` a la bonne syntaxe (`%d` et `&`).

Boucle 0, 75 pt

0,5 il y a exactement une boucle.

+0,5 c'est un `while`.


+0,25 la condition de boucle teste le débordement de tableau (condition du `for` ou du `while`).

Extras +0,25 pt le programme utilise une variable booléenne, et les constantes symboliques `TRUE` et `FALSE`.

+ 0,25 Pour la présence des deux cas d'affichage (si trouvé si pas trouvé) avec affichage de l'indice et non de la valeur cherchée.

4 Fractions (5 points)

Question H. Une fraction $\frac{p}{q}$ est définie par deux entiers p et q . Le nombre q appelé dénominateur est nécessairement non nul et sera toujours positif, le nombre p , appelé numérateur, peut être négatif ou nul. Définir un type utilisateur pour les fractions (sans tenir compte des questions de signe qui n'ont d'importance que pour l'affichage).

 1 pt
9 min


Correction.

```
struct fraction_s
{
    int num; /* numérateur */
    int den; /* dénominateur */
}; /* <- n'oublie pas le point-virgule ! */
```

Barème. Soit tout juste (noms arbitraires) soit faux.

1 pt même si oublie du point-virgule final, ou si `typedef` zéro sinon

Question I. Déclarer et définir une fonction `multiplier_fractions` qui prend deux fractions en argument et renvoie la fraction obtenue par multiplication des deux fractions (ne pas chercher à simplifier la fraction obtenue). Répondre en faisant bien apparaître d'une part la déclaration, d'autre part la définition.

 1.5 pt
13 min

Correction. Déclaration :

```
struct fraction_s multiplier_fractions(struct fraction_s x,
                                     struct fraction_s y);
```

Définition :


```

struct fraction_s multiplier_fractions(struct fraction_s x,
                                      struct fraction_s y)
{
    struct fraction_s res;
    res.num = x.num * y.num; /* produit des numérateurs "sur" */
    res.den = x.den * y.den; /* produit des dénominateurs */
    return res;
}

```

Barème. Maximum 1,5 pt. Sinon :

0,5 pour la déclaration exacte.

1 pour la définition.

0,5 si la définition retourne effectivement un point mais qui n'a pas la bonne valeur.

Question J. Même question pour la somme de deux fractions `additionner_fractions`.  1 pt
9 min

Correction. Déclaration :

```

struct fraction_s additionner_fractions(struct fraction_s x,
                                       struct fraction_s y);

```

Définition :

```

struct fraction_s additionner_fractions(struct fraction_s x,
                                       struct fraction_s y)
{
    struct fraction_s res;
    res.num = x.num * y.den + y.num * x.den;
    res.den = x.den * y.den; /* produit des dénominateurs */
    return res;
}

```

Barème. Maximum 1 pt.

0,25 pour la déclaration exacte.

0,75 pour la définition, dont [pt 0,5] pour un dénominateur exact.


Question K. Déclarer et définir une procédure affichant une fraction passée en argument exactement comme dans l'exemple suivant où les fractions $\frac{34}{26}$, $\frac{-34}{26}$, $\frac{34}{1}$, $\frac{0}{1}$ sont affichées tour à tour :

```

34/26
-34/26
34
0

```

Attention à bien respecter les deux derniers affichages.

 1.5 pt
13 min

Correction. Déclaration :

```
void afficher_fraction(struct fraction_s x);
```

Définition :

```
void afficher_fraction(struct fraction_s x)
{
    if (x.den == 1)
    {
        printf("%d\n", x.num);
    }
    else
    {
        printf("%d/%d\n", x.num, x.den);
    }
}
```

Barème. *Maximum* 1,5 pt.

0,5 pour la déclaration exacte.

1 pour la définition, dont 0,5 pt pour la simple présence d'un if.


5 Fonctions (3 points)


Question L. Déclarer et définir :


1. une fonction `valeur_absolue` qui prend en entrée un argument réel et retourne sa valeur absolue ;
2. Une procédure `afficher_ligne` qui prend en entrée un entier `n` et un caractère `c` et affiche une ligne contenant `n` fois la caractère `c` ;
3. Une fonction `neper` qui prend en entrée un entier `n` et retourne la valeur de la somme suivante :

$$1 + \sum_{k=1}^{k=n} \frac{1}{k!}.$$

Vous pouvez faire appel à une fonction `int factorielle(int n)` calculant la factorielle de son argument.

 1 pt
9 min

 1 pt
9 min

 1 pt
9 min

Correction. Déclarations :

```
double valeur_absolue(double x);
void afficher_ligne(int n, char c);
double neper(int n);
```

Définitions :

```
double valeur_absolue(double x)
{
    if (x < 0)
    {
        return -x;
    }
    return x;
}
```

```

void afficher_ligne(int n, char c)
{
    int i;
    for (i = 0; i < n; i = i + 1)
    {
        printf("%c", c);
    }
    printf("\n"); /* optionnel */
}

double neper(int n) /* récursive */
{
    if (n > 1)
    {
        return neper(n - 1) + 1.0/factorielle(n);
    }
    return 1.0;
}
/* alternative plus classique */
double neper(int n)
{
    double somme;
    int i;
    for (i = 1; i <= n; i = i + 1)
    {
        somme = somme + 1.0/factorielle(i);
    }
    return somme;
}


```

Barème. On note 0,5 pt par déclaration exacte et 0,5 pt par définition correcte (en dehors de la question de typage) en admettant comme erreurs l'oubli de l'utilisation de %c pour l'affichage d'un caractère, une erreur d'indice à un près dans les boucles, un problème d'arrondi entier.

On ajoute 0,5 pt si il est fait correctement appel à la fonction factorielle, tant que la syntaxe d'appel est correct et quelle que soit les paramètres et l'usage de cet appel.

Bonus

Question bonus. Au choix. Déclarer et définir une fonction qui calcule le pgcd de deux entiers positifs ou nuls. Ou bien, déclarer et définir une procédure `afficher_disque` qui prend en paramètre un entier `rayon` et affiche un disque d'étoiles de ce rayon.

 2 pt
18 min

Correction. Voici trois corrigés, le pgcd récursif, un pgcd itératif (avec boucles au lieu de récursion), la procédure `afficher_disque`. Déclarations :

```

int pgcd_r(int a, int b);
int pgcd_i(int a, int b);
void afficher_disque(int rayon);

```

Définitions :

```

int pgcd_r(int a, int b)
{
    if (a < b)
    {
return pgcd(b, a);
    }
    if (b == 0)
    {
return a;
    }
    return pgcd(b, a % b);
}

int pgcd_i(int a, int b)
{
    int i;
    int min;
    int d;
    /* cas particuliers */
    if (a == 0)
    {
return b;
    }
    if (b == 0)
    {
return a;
    }
    /* min = minimum(a, b) */
    if (a < b)
    {
min = a;
    }
    else
    {
min = b;
    }
    /* diviseurs */
    for (i = 1; i <= min; i = i + 1)
    {
/* i diviseur commun ? */
if ( (0 == a % i) && (0 == b % i) )
{
    d = i;
}
    } /* d plus grand diviseur commun */
    return d;
}

void afficher_disque(int rayon)
{
    /* on balaie le carré contenant le disque, de côté = 2 rayons + 1,
       et on affiche une étoile si la distance au centre est

```

```

        inférieure au rayon, sinon un blanc */
    int i;
    int j;
    for (i = -rayon; i <= rayon; i = i + 1)
    {
for (j = -rayon; j <= rayon; j = j + 1)
{
    if ( (i*i+j*j) <= rayon*rayon )
    {
printf("*");
    }
    else
    {
printf(" ");
    }
}
printf("\n");
    }
}

```