
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.

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
```

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.

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

Question D. Simulez l'exécution du programme figure 1 page 11, en réalisant sa **trace**, comme cela a été vu en TD et en cours.

 4 pt
36 min

Correction. Table 1 page 3.

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`.

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.

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

```

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.

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 }

```

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 ! */

```

Question I. Déclarer et définir une fonction `multiplier_fractions` qui prend deux fractions en argument et renvoie la fraction obtenue par multiplication des deux frac-

tions (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;  
}
```

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;  
}
```

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);
    }
}


```

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;  1 pt
9 min
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`;  1 pt
9 min
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

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 = 1.0;
    int i;
    for (i = 1; i <= n; i = i + 1)
    {
        somme = somme + 1.0/factorielle(i);
    }
    return somme;
}
```

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");  
    }  
}
```

```

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