Eléments de programmation 2 – 1i002 – CORRIGÉ

Partiel du 10 mars 2017 (1h30)
Seule la fiche mémo est autorisée

*Les calculatrices, baladeurs et autres appareils électroniques sont interdits. Les téléphones mobiles doivent être éteints et rangés dans les sacs. Le barème sur 40 points n'a qu'une valeur indicative. Avant de commencer, vérifiez que votre copie contient **11 pages** avec 12 questions.*

1 Cours (8 pts)

Question 1 (3 points)

Pour chacune des déclarations suivantes, dire si elle est juste (*i.e.* si elle ne provoque, ni erreur, ni avertissement à la compilation) ou non. Si non, expliquer pourquoi elle est syntaxiquement fausse ou provoque un avertissement à la compilation et proposer une solution pour la corriger. Une réponse sans proposition de correction sera considérée comme fausse.

```
1  int x = 2;
2  char chaine[] = "Hello";
3  float z = 2,3;
4  int t, int v;
5  float tab[3] = {1.0, 2.0};
6  #define N 3;
```

Solution:

- Ligne 1 : cette déclaration est correcte.
- Ligne 2 : cette déclaration est correcte.
- Ligne 3 : erreur de compilation, car en `C` les nombres en virgule flottante ont leur partie entière séparée par un point de leur partie décimale. Une bonne déclaration est :

  ```
  float z = 2.3;
  ```

- Ligne 4 : il y a une erreur de compilation car dans une déclaration de deux valeurs de même type, il n'est pas nécessaire de répéter le type. Une bonne déclaration est :

  ```
  int t, v;
  ```

- Ligne 5 : cette déclaration est correcte, mais la troisième valeur du tableau sera indéfinie.
- Ligne 6 : cette déclaration est correcte.

Question 2 (3 points)

Donner le nombre d'itérations minimum qu'il est possible de faire en `C` avec un `for`, un `while` et un `do-while`. Donner un exemple à chaque fois pour illustrer votre réponse.

Solution:

- 0 itération pour un `for`
- 0 itération pour un `while`
- 1 itération pour un `do-while`

Des solutions possibles sont :

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

while (i>100) {
  i++;
}

do {
  i++;
} while (i>100);
```

Question 3 (2 points)

Décrire de manière précise le résultat de l'exécution de ce programme suivant en justifiant votre réponse.

```
#include <stdio.h>

int main () {
  int i;
  int f[6]={1, 2};
  for (i=2; i<=6; i++) {
    f[i] = f[i-1]+f[i-2];
    printf("%d ", f[i]);
  }

  return 0;
}
```

Solution: Le programme affiche : `3 5 8 13` puis lorsque $i = 6$ accède à une case du tableau f en dehors des limites de celui ci. Ceci n'est pas autorisé en `C`.

2 Exercices (17 pts)

Question 4 (3 points)

Une fraction A/B est définie par 2 entiers A et B. Nous considèrerons ici que A et B sont deux entiers non nuls.

Écrire un programme qui, à partir de deux fractions A/B et C/D, calcule la multiplication de ces deux fractions et affiche à l'écran le résultat de ce calcul. Les valeurs de A, B, C et D sont définies par des directives **`#define`**. Par exemple, pour A = 2, B = 5, C = 3 et D = 4, le programme doit afficher : "Le produit de 2/5 par 3/4 est 6/20". Le code doit bien sûr être indépendant des valeurs de A, B, C et D.

Solution:

```
#include <stdio.h>
#define A 3
```

```c
#define B 4
#define C 7
#define D 9
int main() {
  int numerateur = A*C;
  int denominateur = B*D;
  printf("Le produit de %d/%d par %d/%d est %d/%d\n", A, B, C, D,
     numerateur, denominateur);
  return 0;
}
```

Question 5 (3 points)

Donald sait planter les choux. Il plante trois choux dans une première rangée. Il augmente de deux choux d'une rangée à l'autre. Écrire un programme `C` qui affiche le nombre total de choux plantés par Donald s'il fait en tout 49 rangées. Vous utiliserez des directives **`#define`** pour définir les constantes du programme.

Solution:

```c
#include <stdio.h>
#define NB_CHOUX_INITIAL 3
#define NB_RANGS 49

int main() {
   int nb_choux = NB_CHOUX_INITIAL;
   int i, total = nb_choux;
   for (i=2; i<=NB_RANGS; i++) {
      nb_choux = nb_choux+2;
      total = total + nb_choux;
   }
   printf("Sur %d rangees, Donald a plante %d choux.\n", NB_RANGS, total);
   return 0;
}
```

Question 6 (4 points)

Écrire un programme qui calcule la fonction $w(n) = \frac{1*3*...*(2n+1)}{2*4*...*(2n)}$ avec $n >= 1$. Par exemple $w(3) = 2.1875$ $(= \frac{1*3*5*7}{2*4*6} = \frac{105}{48})$. Le programme demandera à l'utilisateur d'entrer la valeur de n au clavier en vérifiant qu'il entre bien une valeur strictement positive (*i.e.* saisie avec vérification, donc on lui demande de saisir une valeur jusqu'à ce que celle-ci soit valide), puis affichera à l'écran les valeurs de n et de la fonction calculée.

Solution: Ici il faut penser à faire une saisie avec vérification (boucle **`do...while`**) pour récupérer une valeur strictement positive pour n. Ensuite, il suffit de faire une boucle pour calculer la fonction w. Il ne faut pas oublier d'initialiser les variables qui contiennent le numérateur et le dénominateur de la fonction à 1 (élément neutre de la multiplication).

```c
#include <stdio.h>
```

```c
int main() {
  float w = 0.0;
  float num = 1.0, den = 1.0;
  int n, i;

  do {
    printf("\n Veuillez entrer une valeur strictement positive pour
       n?");
    scanf("%d",&n);
  } while (n <= 0);

  for (i=1; i<= n; i++) {
    num = num*(2*i+1);
    den = den*(2*i);
  }
  w = num/den;

  printf("\nLa fonction w pour la valeur %d est egale a %f\n", n, w)
     ;

  return 0;
}
```

Question 7 (3 points)

Écrire un programme qui remplit un tableau de N entiers en tirant aléatoirement le contenu de chaque case parmi les valeurs 2, 3 et 5. N sera défini par un **#define**. Le programme affichera ensuite le nombre de 2, de 3 et de 5 contenus dans ce tableau.

Solution: Il faut utiliser les fonctions de tirage aléatoire, et donc inclure les bibliothèques nécessaires.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 10

int main() {
  int tab[N];
  int i;
  int tirage;
  int nb2 = 0, nb3 = 0, nb5 = 0;

  srand(time(NULL));

  for (i=0;i<N;i++) {
    tirage = rand()%3;
    if (tirage == 0)
```

```
      {
        tab[i] = 2;
        nb2++;
      }
    else
      if (tirage == 1)
        {
          tab[i] = 3;
          nb3++;
        }
      else
        {
          tab[i] = 5;
          nb5++;
        }
  }
  printf("Nombre de 2: %d\n", nb2);
  printf("Nombre de 3: %d\n", nb3);
  printf("Nombre de 5: %d\n", nb5);

  return 0;
}
```

Question 8 (4 points)

On considère des chaînes de caractères formées des deux caractères parenthèses ouvrante et fermante : `(` et `)`. On veut écrire un programme qui détermine si une chaîne de caractères est bien parenthésée comme : `(()())` ou `(((())))` ou encore `()()(((())))` ou mal parenthésée comme : `())` ou `)(` ou encore `(((()))`. Dans le cas d'une chaîne de caractères mal parenthésée, il y a au moins une parenthèse ouvrante qui ne correspond à aucune parenthèse fermante ou une parenthèse fermante qui ne correspond à aucune parenthèse ouvrante.

Pour déterminer cela, on parcourt la chaîne de caractères de gauche à droite et on compte les parenthèses. On ajoute +1 chaque fois que l'on trouve une parenthèse ouvrante et -1 chaque fois que l'on trouve une parenthèse fermante. Le programme doit s'arrêter dès que le compteur devient négatif ou lorsque l'on atteint la fin de la chaîne de caractères.

Votre programme écrira à la fin la chaîne de caractères et si cette chaîne de caractères est bien ou mal parenthésée. Les caractères autres que les parenthèses seront ignorés par votre programme.

Solution: Afin d'éviter de parcourir tout le tableau pour rien, il faut une boucle dont on puisse sortir dès que le compteur devient négatif. La boucle **`while`** est la mieux adaptée. Voici une solution déterminant la fin de la chaîne par le caractère de fin de chaîne :

```
#include <stdio.h>

int main() {
  char expression[] = "(()())";
  int compteur = 0;
  int i=0;
```

```
  while ((expression[i]!='\0') && (compteur>=0)) {
    if (expression[i] == '(')
      {
        compteur++;
      }
    if (expression[i] == ')')
      {
        compteur--;
      }
    i++;
  }

  if (compteur == 0) {
    printf("La chaine %s est bien parenthesee\n", expression);
  }
  else {
    printf("Chaine %s est mal parenthesee\n", expression);
  }

  return 0;
}
```

La solution déterminant la fin de la chaîne par un comptage manuel du nombre de caractères sera pénalisée :

```
#include <stdio.h>
#define N 10

int main() {
  char expression[N] = "(()())";
  int compteur = 0;
  int i=0;
  while ((i<N) && (compteur>=0)) {
    if (expression[i] == '(')
      {
        compteur++;
      }
    if (expression[i] == ')')
      {
        compteur--;
      }
    i++;
  }

  if (compteur == 0) {
    printf("La chaine %s est bien parenthesee\n", expression);
  }
  else {
    printf("Chaine %s est mal parenthesee\n", expression);
```

```
  }

  return 0;
}
```

3 Manipulation de boucles (7 pts)

Question 9 (3 points)

Soient deux tableaux de nombres de type **float** de tailles respectives N et M avec N et M >= 1. On veut calculer la somme de l'ensemble des valeurs obtenues en multipliant chaque élement du premier tableau par chaque élément du deuxième tableau.

Écrire le programme qui affiche ce total. Vous utiliserez une itération de type **for** et vous initialiserez les deux tableaux avec les valeurs de votre choix.

Solution:

```
#include <stdio.h>
#define N 5
#define M 3
int main() {
  float t1[N] = {5.5, 1.2, 7.2, 2.8, 3.1};
  float t2[M] = {3.4, 10.3, 34.1};
  float resultat = 0;
  int i, j;
  for (i=0; i<N; i++) {
    for (j=0; j<M; j++) {
      resultat = resultat + t1[i]*t2[j];
    }
  }
  printf("Resultat : %f\n", resultat);
  return 0;
}
```

Question 10 (4 points)

On fait une simulation de jeu de combat entre deux joueurs. Chacun des deux joueurs dispose d'un nombre de points de vie initial `PVINIT`. À chaque tour de jeu, chacun des joueurs inflige à l'autre un nombre de points de dégâts tiré aléatoirement par le lancer d'un dé à 6 faces : ces points de dégâts sont retirés des points de vie de son adversaire.

Le combat se termine lorsqu'au moins un des deux joueurs a un nombre de points de vie inférieur ou égal à zéro (il peut y avoir match nul).

Écrire un programme qui effectue un combat où, à chaque tour, on affiche le numéro du tour, le nombre de points de vie des deux joueurs au début du tour et le nombre de points de dégâts infligés par chaque joueur à son adversaire. À la fin du dernier tour, le programme affiche le joueur gagnant et le nombre de tours joués.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define PVINIT 2

int main() {

  int pvjoueur1 = PVINIT, pvjoueur2 = PVINIT;
  int pvdegat1, pvdegat2;
  int tour = 1;
  srand(time(NULL));
  do
    {
      printf("Tour␣%d\n", tour);
      printf("Le␣joueur␣1␣a␣%d␣points␣de␣vie.\n", pvjoueur1);
      printf("Le␣joueur␣2␣a␣%d␣points␣de␣vie.\n", pvjoueur2);
      pvdegat1 = 1+ rand()%6;
      pvdegat2 = 1+ rand()%6;
      pvjoueur1 = pvjoueur1 - pvdegat1;
      printf("Le␣joueur␣1␣fait␣%d␣points␣de␣degats.\n", pvdegat1);
      pvjoueur2 = pvjoueur2 - pvdegat2;
      printf("Le␣joueur␣2␣fait␣%d␣points␣de␣degats.\n", pvdegat2);
      tour++;
    }
  while ((pvjoueur1 >0)&&(pvjoueur2>0));
  if ((pvjoueur1<=0)&&(pvjoueur2<=0))
    {
      printf("Match␣nul␣en␣%d␣coups.\n", tour-1);
    }
  if ((pvjoueur1>0) && (pvjoueur2<=0))
    {
      printf("Le␣joueur␣1␣a␣gagne␣en␣%d␣coups.\n", tour-1);
    }
  if ((pvjoueur1<=0) && (pvjoueur2>0))
    {
      printf("Le␣joueur␣2␣a␣gagne␣en␣%d␣coups.\n", tour-1);
    }
  return 0;
}
```

4 Problème (8 pts)

Question 11 (4 points)

Un carré magique d'ordre n est composé de n^2 entiers strictement positifs rangés dans un tableau de

$n \times n$ cases. Les valeurs sont placées de sorte que les sommes des nombres sur chaque ligne, dans chaque colonnne et sur chaque diagonale soient toutes égales. La valeur unique de ces sommes est appelée constante magique.
Écrire un programme qui teste si un tableau est un carré magique. Si oui, le programme affiche la constante magique. Sinon, dès qu'il détecte que le tableau n'est pas un carré magique, il doit se terminer en affichant un message.

Solution:

```
#include <stdio.h>
#define N 3

int main() {
   int t[N][N] = {
      {4, 9, 2},
      {3, 5, 7},
      {8, 1, 6}};

   int i, j;
   int constante ;
   int somme = 0;

   // Calcul de la somme de la premiere diagonale
   for (i=0; i<N; i++) {
      somme = somme + t[i][i];
   }

   constante = somme;
   somme = 0;
   // Calcul sur la 2ieme diagonale
   for (i=0; i<N; i++) {
      somme = somme + t[i][N-i-1];
   }

   i = 0;
   while ((i < N) && (somme == constante)) {
      // Verification de la somme sur la ligne i
      somme = 0;
      for (j = 0; j<N; j++) {
         somme = somme + t[i][j];
      }
      i++;
   }

   j = 0;
   while ((j < N) && (somme == constante)) {
      // Verification de la somme sur la colonne j
      somme = 0;
      for (i = 0; i<N; i++) {
         somme = somme + t[i][j];
```

```
      }
      j++;
   }
   if (somme == constante) {
      printf("Le carre est magique avec une constante magique = %d\n",
         constante);
   }
   else {
      printf("Carre non magique.\n");
   }
   return 0;
}
```

Question 12 (4 points)

On souhaite écrire un programme qui fait l'affichage graphique des valeurs contenues dans un carré (magique ou non) en utilisant la bibliothèque CINI. On se limitera à des carrés contenant des entiers inférieurs à 100. On affectera à l'affichage de chaque valeur un rectangle de taille 30 pixels de large par 25 pixels de haut.

Pour afficher le contenu d'une case, on utilisera la fonction `CINI_draw_string` qui affiche une chaîne de caractères. Pour convertir un entier en chaîne de caractères, on utilisera la fonction `sprintf` de la bibliothèque `stdio.h` qui copie dans un tableau de $k+1$ caractères déclaré préalablement la chaîne correspondant à un nombre d'au plus k chiffres. Par exemple :

```
char str[3];
int n = 99;
sprintf(str, "%d", n);
```

copie la chaîne `"99"` dans `str`.

La taille de la fenêtre d'affichage correspondra à la taille du carré plus une bordure de `MARGE` pixels autour.

Il n'est pas nécessaire de dessiner le quadrillage mais juste les valeurs du carré.

Solution:

```
#include <stdio.h>
#include <cini.h>
#define N 3
#define MARGE 20
#define LARG 30
#define HAUT 25

int main() {
  int t[N][N] = {
    {4, 9, 2},
    {3, 5, 7},
    {8, 1, 6}};

  char str[3];
  int i, j;
```

```
  CINI_open_window(2*MARGE+N*LARG, 2*MARGE+N*HAUT, "Carre␣magique");
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        sprintf(str, "%d", t[j][i]);
        CINI_draw_string(i*LARG+MARGE, j*HAUT+MARGE, "yellow", str);
    }
  }
  CINI_loop();
  return 0;
}
```