CS-119(a) – ICC-C Série 2

2024-02-27

Exo1 Décomposition en base 10

L'utilisateur rentre un nombre entier à 3 chiffres. Lisez-le avec la fonction `scanf` et donnez sa décomposition en base 10. Par exemple, si on rentre 142, on devrait voir la sortie :

```
centaines: 1, dizaines: 4, unités: 2
```

Solution de l'exercice 1

```c
int number;
printf("Rentrez un entier à 3 chiffres: ");
scanf("%d", &number);
printf("centaines: %d, dizaines: %d, unités: %d\n",
       number / 100 % 10,
       number / 10 % 10,
       number % 10);
```

Exo2 Moyenne

Bob aimerait calculer la moyenne d'un vecteur de 5 nombres. Il utilise ce code :

```c
int vec[] = {1, 2, 3, 4, 5};
printf("La moyenne est %d\n",
       1 / 5 * (vec[0] + vec[1] + vec[2] + vec[3] + vec[4]));
```

Pourtant, le code ne semble pas bien fonctionner. Pourquoi ? Qu'affiche-t-il ? Il essaye une autre version :

```c
int vec[] = {1, 2, 3, 4, 5};
printf("La moyenne est %d\n",
       vec[0]/5 + vec[1]/5 + vec[2]/5 + vec[3]/5 + vec[4]/5);
```

Le résultat a changé, mais ce n'est toujours pas ça. Pourquoi ? Qu'affiche-t-il ?

"Cette fois c'est la bonne !" se dit Bob. Et il écrit :

```
int vec[] = {1, 2, 3, 4, 5};
int somme = vec[0] + vec[1] + vec[2] + vec[3] + vec[4];
double moyenne = somme / 5;
printf("La moyenne est %g\n", moyenne);
```

Ceci affiche effectivement la bonne moyenne. Est-ce que cette dernière version est juste pour autant ?

Donnez un meilleur code pour calculer une moyenne. Lisez les 5 nombres depuis stdin avec scanf.

Solution de l'exercice 2

La première variante ne fonctionne pas, car `1/5` est `0` à cause de la division entière, du coup le code affiche `0`.

La deuxième variante a le même souci, mais le code affiche `1` car les premiers quatre termes de la somme `1/5`, ..., `4/5` sont `0`, sauf le dernier élément qui est `5/5`, donc `1`.

La variante finale du code est quand même fausse, car elle utilise toujours la division entière ! Même si la variable `moyenne` est de type `double`, elle prend la valeur de la division entière `(1+2+3+4+5)/5` qui est `15/5`, et donc `3`. Par contre, si on remplaçait la dernière valeur `vec[4] = 6`, le résultat serait aussi `3`, et non pas `3.2`.

Une bonne solution serait :

```
int vec[5];
scanf("%d %d %d %d %d",
      &vec[0], &vec[1], &vec[2], &vec[3], &vec[4]);
int somme = vec[0] + vec[1] + vec[2] + vec[3] + vec[4];
double moyenne = somme / 5.0;
printf("La moyenne est %g\n", moyenne);
```

Exo3 Affichage LCD

On aimerait écrire des nombres composés de quatre chiffres en affichage à segments. Affichez `2`, `0`, `2`, et `4` de cette manière. On devrait voir apparaitre :

```
 _   _   _
 _| | |  _| |_|
|_  |_| |_    |
```

Suggestion : définissez des constantes de type tableau de string `char[4][4]` qui contiennent les formes des chiffres et ensuite combinez-les pour écrire des chiffres en affichage à segments :

```
  char four[4][4] = {
    "   ",
    "|_|",
    "  |",
    "   "
  };
```

Solution de l'exercice 3

```
char zero[4][4] =
    {" _ ",
     "| |",
     "|_|",
     "   "};

char two[4][4] =
    {" _ ",
     " _|",
     "|_ ",
     "   "};

char four[4][4] =
    {"   ",
     "|_|",
     "  |",
     "   "};

printf(
    "%s %s %s %s\n"
    "%s %s %s %s\n"
```

```
    "%s %s %s %s\n"
    "%s %s %s %s\n",
    two[0], zero[0], two[0], four[0],
    two[1], zero[1], two[1], four[1],
    two[2], zero[2], two[2], four[2],
    two[3], zero[3], two[3], four[3]);
```

Exo4 Affichage LCD (suite)

Lisez un chiffre depuis l'entrée standard avec `scanf` et écrivez-le en affichage à segments.

Suggestion : définissez un vecteur de constantes de type tableau de tableaux de string qui correspondent aux chiffres de `0` à `9`. Utilisez le chiffre que vous lisez depuis l'entrée standard comme indice du tableau ainsi défini.

```
char lcd[10][4][4] = ...;
```

Solution de l'exercice 4

```
char lcd[10][4][4] = {
        {" _ ",
         "| |",
         "|_|",
         "   "},
        {"   ",
         "  |",
         "  |",
         "   "},
        {" _ ",
         "  |",
         "|_ ",
         "   "},
        {" _ ",
         "  |",
         " _|",
         "   "},
        {"   ",
         "|_|",
         "  |",
         "   "},
        {" _ ",
         "|_ ",
         " _|",
```

```
          "   "},
         {" _ ",
          "|_ ",
          "|_|",
          "   "},
         {" _ ",
          "  |",
          "  |",
          "   "},
         {" _ ",
          "|_|",
          "|_|",
          "   "},
         {" _ ",
          "|_|",
          " _|",
          "   "}};
int chiffre;
scanf("%d", &chiffre);
printf("%s\n"
       "%s\n"
       "%s\n"
       "%s\n",
       lcd[chiffre % 10][0],
       lcd[chiffre % 10][1],
       lcd[chiffre % 10][2],
       lcd[chiffre % 10][3]);
```

Exo5 Horologe LCD

On aimerait écrire l'heure et les minutes en affichage à segments. Par exemple :

```
   _     _   _
| |_  .  _| |_|
|  _| . |_  |_|
```

Le code pour obtenir l'heure et les minutes de l'horologe de la machine est donné ci-dessous :

```
#include <time.h>

int main() {
  ...
```

```
    time_t t = time(NULL);
    int heure = localtime(&t)->tm_hour;
    int minute = localtime(&t)->tm_min;
    ...
  }
```

Solution de l'exercice 5

On utilise le vecteur `lcd` de la réponse précédente.

```
#include <time.h>

int main() {
  time_t t = time(NULL);
  int heure = localtime(&t)->tm_hour;
  int minute = localtime(&t)->tm_min;

  int heure1 = heure / 10, heure2 = heure % 10;
  int minute1 = minute / 10, minute2 = minute % 10;

  printf(
      "%s %s   %s %s\n"
      "%s %s . %s %s\n"
      "%s %s . %s %s\n"
      "%s %s   %s %s\n",
      lcd[heure1][0], lcd[heure2][0], lcd[minute1][0], lcd[minute2][0],
      lcd[heure1][1], lcd[heure2][1], lcd[minute1][1], lcd[minute2][1],
      lcd[heure1][2], lcd[heure2][2], lcd[minute1][2], lcd[minute2][2],
      lcd[heure1][3], lcd[heure2][3], lcd[minute1][3], lcd[minute2][3]);
}
```