ECUE «Introduction à la programmation » - Session 2

12 juin 2014 - Bruno Bouzy

sans document - durée 1 heure 30

CORRIGE

Exercice 1 (2 points)

Ecrire un programme `exo1.c` permettant à l'utilisateur d'entrer un nombre de victoires et un nombre de défaites et affichant le pourcentage de victoires. En particulier, la sortie du programme doit correspondre à l'exécution ci-dessous. On suppose que l'utilisateur entre des valeurs strictement positives. On affichera le pourcentage avec un chiffre après la virgule. Les entrées clavier sont indiquées en caractères gras.

```
nombre de victoires ? 15
nombre de defaites ? 25
 % victoires = 37.5
```

```
#include <stdio.h>
int main() {
  float v, d;
  printf("nombre de victoires ? ");
  scanf("%f", &v);
  printf("nombre de victoires = %.0f\n", v);
  printf("nombre de defaites ? ");
  scanf("%f", &d);
  printf("nombre de defaites = %.0f\n", d);
  printf("pourcentage victoires = %.1f\n", 100*v/(v+d));
  return (0);
}
```

2 points.

Exercice 2 (2 points)

Donner la sortie du programme suivant.

```
int main() {
  int a = 5;  int * p = &a;  int b = *p;
                printf("1: a = %d, b = %d, *p = %d.\n", a, b, *p);
  a *= 3;       printf("2: a = %d, b = %d, *p = %d.\n", a, b, *p);
  b += 2;       printf("3: a = %d, b = %d, *p = %d.\n", a, b, *p);
  int * q = &b; printf("4: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  *q *= (*p)++; printf("5: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  *q += ++(*p); printf("6: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  p = q;        printf("7: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  q = &a;       printf("8: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  return(0);
}
```

```
1: a = 5, b = 5, *p = 5.
2: a = 15, b = 5, *p = 15.
3: a = 15, b = 7, *p = 15.
4: a = 15, b = 7, *p = 15, *q = 7.
5: a = 16, b = 105, *p = 16, *q = 105.
6: a = 17, b = 122, *p = 17, *q = 122.
7: a = 17, b = 122, *p = 122, *q = 122.
8: a = 17, b = 122, *p = 122, *q = 17.
```

0.25 point par ligne correcte.

Exercice 3 (5 points)

On considère les suites de nombres réels U_n et V_n définies de la manière suivante:

$U_0=1$ $V_0=1$ $U_{n+1}=(1 - V_n^2)/2$ $V_{n+1}=(U_n+V_n)^2/2$

1) a) Quelles sont les valeurs de U_1 et V_1 ?

$U_1=0$ $V_1=2$
0.25 point

1) b) Donner la sortie de:

```
float u=1,v=1; u=(1-v*v)/2; v=(u+v)*(u+v)/2; printf("u=%.2f, v=%.2f\n", u, v);
```

```
u=0.00, v=0.50
0.25 point
```

1) c) La sortie de 1) b) est-elle cohérente avec la définition de U_n et V_n ?

```
Non
0.25 point
```

1) d) Modifier le traitement de 1) b) pour qu'il soit cohérent avec la définition de U_n et V_n.

```
Avec une variable intermédiaire :
    float u_save = u;
    u = (1 - v*v)/2;
    v = (u_save+v)*(u_save+v)/2;
1 point
```

2) Ecrire une fonction `void deUVaUV(float * u, float * v)` prenant `u, v` en entrée et les valorisant en sortie selon 1) d).

```
void deUVaUV(float * u, float * v)
{
  float u_save = *u;
  *u = (1 - *v**v)/2;
  *v = (u_save+*v)*(u_save+*v)/2;
}
1.25 point
```

3) Ecrire un programme `main` affichant les `n` premiers termes des suites U_n et V_n avec 6 chiffres après la virgule. On utilisera `deUVaUV` et `n=5`.

```
#include <stdio.h>
#define N_ITERATIONS 5
…
int main()
{
  int n=0;
  float u=1, v=1;
  for (n=0; n<=N_ITERATIONS; n++) {
    printf ("U%d= %.6f, V%d=%.6f\n", n, u, n, v);
    deUVaUV(&u, &v);
  }
  return 0;
}
2 points
```

Exercice 4 (11 points)

Dans cet exercice, on veut programmer la mise à jour d'une ligne du jeu de `2048` lorsque le joueur joue vers la gauche. Une ligne contient `4` nombres. Un nombre est soit `0`, soit un entier puissance de `2`. Quand le joueur effectue l'action à gauche sur une « ligne entrée », cela transforme la ligne en une « ligne sortie » avec un décalage des nombres non nuls vers la gauche, suivi d'une fusion éventuelle de deux nombres voisins égaux, et enfin suivi d'un autre éventuel décalage vers la gauche. Cf exemples ci-dessous.

« ligne entrée »:	« ligne sortie »:	Commentaire :
`(0 0 2 0)`	`(2 0 0 0)`	le `2` est venu à gauche. Un `0` est vu comme du vide.
`(0 0 0 8)`	`(8 0 0 0)`	le `8` est venu à gauche. Un `0` est vu comme du vide.
`(4 0 0 0)`	`(4 0 0 0)`	le `4` reste à gauche.
`(0 2 0 4)`	`(2 4 0 0)`	le `2` et le `4` sont venus à gauche.
`(2 32 4 16)`	`(2 32 4 16)`	les nombres n'ont pas bougé.
`(16 16 0 2)`	`(32 2 0 0)`	les deux `16` ont fusionné en `32`, le `2` est venu à gauche.
`(16 0 16 2)`	`(32 2 0 0)`	les deux `16` ont fusionné en `32`, le `2` est venu à gauche.
`(4 4 4 0)`	`(8 4 0 0)`	les deux `4` ont fusionné en `8`, le dernier `4` est venu à gauche.
`(4 4 8 0)`	`(8 8 0 0)`	les deux `4` ont fusionné en `8`, le dernier `8` est venu à gauche.
`(4 4 4 4)`	`(8 8 0 0)`	les deux 1ers `4` ont fusionné en `8`, les deux derniers `4` aussi.
`(1024 0 0 1024)`	`(2048 0 0 0)`	les deux `1024` ont fusionné en `2048`.

On représente la ligne par un tableau de quatre entiers `int`. On va programmer deux fonctions de base : `decalageGauche(int * tab)` et `fusionGauche(int * tab)`. Ces fonctions seront utilisées par la fonction `miseAJour(int * tab)` appelée par le `main`.

1) Programmer une fonction `int zeroOuPuissanceDe(int n, int p)` retournant `1` si `n` est égal à `0` ou à une puissance de `p`, et retournant `0` sinon.

```
int zeroOuPuissanceDe(int n, int p) {
  if (n==0) return 1;
  if (p==0) return 0;
  do {
    if (n==p) return 1;
    int r = n % p;
    if (r!=0) return 0;
    n = n / p;
  } while (1);
}
```
2 points

2) Programmer une procédure d'initialisation `void init(int * tab)`. Cette procédure remplit le tableau avec des nombres entrés au clavier. Avec la fonction `zeroOuPuissanceDe`, elle vérifie que les nombres entrés sont soit un `0` soit une puissance de `2`.

```
void init(int * tab) {
  int i;
  for (i=0; i<TAILLE; i++) {
    do {
      printf("tab[%d] ? (0 ou puissance de 2) ", i);
      scanf("%d", tab+i);
    } while (zeroOuPuissanceDe(tab[i], 2)==0);
    printf("tab[%d] = %d\n", i, *(tab+i));
   }
}
```
2 points

3) Programmer une procédure d'affichage à l'écran `void affichage(int * tab)`. Elle respectera le format des exemples ci-dessus avec quatre nombres entiers entre parenthèses.

```
void affiche(int * tab) {
  int i;
  printf("( ");
  for (i=0; i<TAILLE; i++) printf("%d ", *(tab+i));
  printf(")\n");
}
```

1 point

4) Programmer la procédure `void decalageGauche(int * tab)`.

```
void decaleGauche(int * tab) {
  int i, j;
  for (i=0; i<3; i++) {
    int j=i;
    while ((j<TAILLE) && (tab[j]==0)) j++; // indice de 1ere case non zero
    if (j==TAILLE) return;
    int k;
    for (k=i; k<TAILLE; k++) {
      if (k+j-i<TAILLE) tab[k] = tab[k+j-i];
      else tab[k] = 0;
    }
  }
}
```

2.5 points

5) Programmer la procédure `void fusionGauche(int * tab)`.

```
void fusionneGauche(int * tab) {
  int i;
  for (i=0; i<TAILLE; i++) {
    if (tab[i]==tab[i+1]) {
      tab[i] += tab[i];
      tab[i+1] = 0;
    }
  }
}
```

1.5 point

6) Programmer la procédure `void miseAJour(int * tab)`. Elle transforme le tableau en respectant les exemples de transformations « ligne entrée » et « ligne sortie » ci-dessus. Elle utilisera des appels convenables à `decalageGauche` et à `fusionGauche`. Elle affiche la ligne après chaque appel à `decalageGauche` et à `fusionGauche`.

```
void miseAJour(int * tab) {
  printf("2048 avant 1er decalage\n");
  decaleGauche(tab);
  affiche(tab);
  printf("2048 avant fusion\n");
  fusionneGauche(tab);
  affiche(tab);
  printf("2048 avant 2eme decalage\n");
  decaleGauche(tab);
}
```

1 point

7) Programmer le `main`. Le `main` contient la déclaration du tableau `ligne`. Il initialise la ligne avec `init`, affiche la ligne avec `affichage`, et met à jour la ligne avec `miseAJour`.

```
#include <stdio.h>
#define TAILLE 4
...
int main()
{
  printf("2048 debut:\n");
  int ligne[TAILLE];
  init(ligne);
  affiche(ligne);
  // printf("2048 avant mise a jour\n");
  miseAJour(ligne);
  // printf("2048 apres mise a jour\n");
  affiche(ligne);
  printf("2048 fin.\n");
  return 0;
}
```

1 point