ECUE «Introduction à la programmation »

Contrôle continu n°3 – 10 janvier 2013
sans document - durée 1 heure 30

CORRIGE

Exercice 1 (2 points)

Avec des boucles `for`, écrire un programme affichant toutes les solutions de l'équation `(1)` :

`A x B = C` (1)

où `A`, `B` et `C` sont trois chiffres compris entre `1` et `9` tous distincts.

2 points

```
#include <stdio.h>
#define TAILLE 9
int main() {
  int a, b, c;
  for (a=1; a<=TAILLE; a++) {
    for (b=1; b<=TAILLE; b++) {
      if (b!=a)
     for (c=1; c<=TAILLE; c++) {
       if ((c!=a) && (c!=b) && (a*b == c))
           printf(" a=%d, b=%d, c=%d\n", a, b, c);
     }
    }
  }
  return 0;
}
```

Exercice 2 (6 points)

1) Ecrire une fonction `int deIntervalleANombre(int a, int b)` demandant à l'utilisateur un nombre entier appartenant à l'intervalle `[a,b]` et retournant ce nombre. La fonction demande répétitivement le nombre à l'utilisateur tant que le nombre n'appartient pas à `[a,b]`.

2 points

```
int deIntervalleANombre(int a, int b) {
  int x;
  do {
    printf("x ? (%d<=x<=%d) ", a, b);
    scanf("%d", &x);
  } while (x<a || x>b);
  return x;
}
```

2) Ecrire une fonction `void tabMaxMin` prenant en entrée un tableau d'entiers `tab` et une longueur `l` de tableau, et donnant en sortie le maximum `max` et le minimum `min` des valeurs du tableau. On choisira adéquatement les types des paramètres de la fonction.

2 points

```
void tabMaxMin(int * tab, int l, int * max, int * min) {
  int i;
  *max = tab[0];
  *min = tab[0];
  for (i=0; i<l; i++) {
    if (*max<tab[i]) *max = tab[i];
    if (*min>tab[i]) *min = tab[i];
  }
}
```

3) Ecrire un programme `main` remplissant un tableau de `10` entiers appartenant à l'intervalle `[0,9]` en utilisant la fonction `deIntervalleANombre`, puis affichant les valeurs du tableau, puis appelant `tabMaxMin` et affichant le maximum et le minimum des valeurs du tableau.

On respectera les entrée-sorties de l'exécution ci-dessous:

```
t[0] : x ? (0<=x<=9) 10
x ? (0<=x<=9) -1
x ? (0<=x<=9) 0
t[1] : x ? (0<=x<=9) 9
t[2] : x ? (0<=x<=9) 5
t[0] = 0 , t[1] = 9 , t[2] = 5 ,
max = 9, min = 0
```

2 points

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

int main() {
  int t[TAILLE], i, max, min;
  for (i=0; i<TAILLE; i++) {
    printf("t[%d] : ", i);
    t[i] = deIntervalleANombre(0,9);
  }
  for (i=0; i<TAILLE; i++) printf("t[%d] = %d, ", i, t[i]);
  printf("\n");
  tabMaxMin(t, TAILLE, &max, &min);
  printf("max = %d, min = %d\n", max, min);
  return 0;
}
```

Exercice 3 (6 points)

On considère la suite U_n de nombres réels définie les équations (2) :

$$U_0 = 0$$
$$U_{n+1} = U_n + 4/(2n+1) \text{ si } n \text{ est pair.} \quad (2)$$
$$U_{n+1} = U_n - 4/(2n+1) \text{ si } n \text{ est impair.}$$

1) Ecrire une fonction `float piLeibniz1(int n)` affichant les `n` premiers termes de la suite U_n et retournant le dernier terme.

```
2 points
float piLeibniz1(int n)
{
  float pi=0;
  int i;
  for (i=0; i<n; i++) {
    printf("i = %2d: ", i);
    if (i%2==0) pi += 4.0/(2*i+1);
    else        pi -= 4.0/(2*i+1);
    printf("pi = %.6f\n", pi);
  }
  return pi;
}
```

2) Ecrire une fonction `float piLeibniz2(float pr, int * n)` affichant les termes de la suite U_n tant que $|U_n - U_{n-1}| > pr$, retournant le dernier terme, et mettant le nombre d'itérations effectuées dans `*n`.

```
3 points
float piLeibnizBis(float precis, int * n)
{
  float pi=0, piErreur;
  int i=0;
  do {
    printf("i = %4d: ", i);
    float piApres;
    if (i%2==0) piApres = pi + 4.0/(2*i+1);
    else        piApres = pi - 4.0/(2*i+1);
    printf("approx pi = %.6f, ", piApres);
    piErreur = piApres - pi;
    if (piErreur<0) piErreur = -piErreur;
    // printf("erreur = %.6f\n", piErreur);
    i++;
    pi = piApres;
  } while (piErreur>precis);
  *n = i;
  return pi;
```

```
}
```

3) Ecrire une fonction `main` appelant la fonction `piLeibniz2` avec une précision `0.01`. et affichant le résultat et le nombre d'itérations effectuées.

1 point
```
int main() {
  int n;
  float pi1 = piLeibnizBis(0.01, &n);
  printf("piLeibniz = %.6f, nIterations = %d\n", pi1, n);
  return 0;
}
```

Exercice 4 (6 points)

On considère les suites A_n , B_n , C_n , D_n , P_n de nombres réels définies de la manière suivante:

$A_0=1$ $\quad B_0=1/2^{1/2}$ $\quad C_0=1/4$ $\quad D_0=1$ (3)

$A_{n+1}=(A_n+B_n)/2$ $\quad B_{n+1}=(A_nB_n)^{1/2}$ $\quad C_{n+1}=C_n-D_n(A_n-B_n)^2/4$ $\quad D_{n+1}=2D_n$ (4)

$P_n = (A_n+B_n)^2/(4C_n)$ (5)

1) a) Donner la sortie du traitement suivant:

```
  float a=1, b, c=0.25, d=1; b=sqrt(0.5);
  printf("a0=%.2f, b0=%.2f, c0=%.2f, d0=%.2f\n", a, b, c, d);
```

0.5 point
```
a0 = 1.00, b0 = 0.71, c0 = 0.25, d0 = 1.00
```

1) b) Donner une approximation de A_1, B_1, C_1, D_1 avec 2 chiffres après la virgule.

0.5 point

$A_1=0.85$ $\quad B_1=0.84$ $\quad C_1=0.23$ $\quad D_1=2.00$

2) Donner la sortie du traitement suivant:

```
  float a=1, b, c=0.25, d=1; b=sqrt(0.5);
  a = (a+b)/2;
  b = sqrt(a*b);
  c = c - d*(a-b)*(a-b)/4;
  d = 2*d;
  printf("a1=%.2f, b1=%.2f, c1=%.2f, d1=%.2f\n", a, b, c, d);
```

0.5 point

a1 = 0.85, **b1 = 0.78, c1 = 0.25**, d1 = 1.00
NB: les valeurs en gras sont fausses car:
b est mis à jour avec la valeur de a1 au lieu de a0.
c est mis à jour avec une valeur erronée de b.

3) Modifier le traitement ci-dessus pour qu'il corresponde à la définition `(4)`.

1 point

```
  float a=1, b, c=0.25, d=1; b=sqrt(0.5);
  float aBis = (a+b)/2;
  float bBis = sqrt(a*b);
  c = c - d*(a-b)*(a-b)/4;
  d = 2*d;
  a = aBis;
  b = bBis;
  printf("a1=%.2f, b1=%.2f, c1=%.2f, d1=%.2f\n", a, b, c, d);
```

4) Ecrire une fonction `void deABCDaABCD(float * a, float * b, float * c, float * d)` prenant `a`, `b`, `c`, `d` en entrée et les valorisant en sortie selon la définition `(4)`.

1 point

```
void deABCDaABCD(float * a, float * b, float * c, float * d) {
  float aBis = (*a+*b)/2;
  float bBis = sqrt(*a**b);
  *c = *c - *d*(*a-*b)*(*a-*b)/4;
  *d = 2**d;
  *a = aBis;
  *b = bBis;
}
```

5) Ecrire une fonction `float deABCaP(float a, float b, float c)` prenant `a`, `b`, `c` en entrée et retournant une valeur correspondant à la définition `(5)`.

0.5 point

```
float deABCaP(float a, float b, float c) {
  return 0.25*(a+b)*(a+b)/c;
}
```

6) Ecrire une fonction `float piBrentSalamin(int n)` affichant les `n` premiers termes de la suite P_n et retournant le dernier terme calculé. On utilisera `deABCDaABCD` et `deABCaP`.

1.5 point

```
float piBrentSalamin(int n) {
  printf("piBrentSalamin:\n");
  float a, b, c, d, pi;
  int i=0; printf("nIterations = %2d: ", i);
  a=1;
  b=sqrtl(0.5);
  c=0.25;
  d = 1;
  pi = deABCaP(a, b, c);
  for (i=1; i<=n; i++) {
    printf("nIterations = %2d: ", i);
    deABCDaABCD(&a, &b, &c, &d);
    pi = deABCaP(a, b, c);
    printf("pi=%8.6f\n", pi);
  }
  return pi;
}
```

7) Ecrire une fonction `main` appelant `piBrentSalamin` avec `3` itérations et affichant le résultat.

0.5 point

```
#include <stdio.h>
#include <math.h>
int main() {
  float pi = piBrentSalamin(3);
  printf("piBrentSalamin = %.6f\n", pi);
  return 0;
}
```

Epilogue

Les suites U_n et P_n des exercices 3 et 4 convergent vers Π. La fonction `piLeibniz2` donne Π avec une précision `0.01` au bout de deux cents itérations et avec une précision `0.000001` au bout de deux millions d'itérations. La fonction `piBrentSalamin` donne Π avec une précision `0.01` en une seule itération et avec une précision `0.000001` en deux itérations. Pour obtenir plus de précision, il faut abandonner le type `float`.

Pour obtenir une précision de 15 chiffres après la virgule, il faut prendre le type `double`. Pour atteindre 18 chiffres, le type `long double`. En 2010, en utilisant des opérateurs arithmétiques dédiés et des ordinateurs spécialisés, on a calculé environ 5000 milliards de décimales de Π.