ECUE «Introduction à la programmation »
Session 2
12 juin 2018
sans document
durée 1 heure 30

CORRIGE

Exercice 1 (3 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
```

```
// victoiresDefaites.c

#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);
}
```

Exercice 2 (5 points)

Donner la sortie du programme ci-dessous. Pour chaque ligne en **caractères gras**, tenir compte de la **couleur de votre copie (bleu, rouge, vert, jaune)** pour **valoriser `a` et `b`** avec les valeurs précisées dans le commentaire de la ligne.

```
#include <stdio.h>
int main()   {
  int a = 3;      // Couleur de la copie : Bleu:3, Rouge:4, Vert:5, Jaune:7
  int * p = &a;
  int b = *p;     printf("1: a = %d, b = %d, *p = %d.\n", a, b, *p);
  a += 4;         // Couleur de la copie : Bleu:4, Rouge:5, Vert:3, Jaune:4
                  printf("2: a = %d, b = %d, *p = %d.\n", a, b, *p);
  b *= 7;         // Couleur de la copie : Bleu:7, Rouge:3, Vert:3, Jaune:3
                  printf("3: a = %d, b = %d, *p = %d.\n", a, b, *p);
  a -= 5;         // Couleur de la copie : Bleu:5, Rouge:7, Vert:4, Jaune:5
                  printf("4: a = %d, b = %d, *p = %d.\n", a, b, *p);
  b /= 3;         // Couleur de la copie : Bleu:3, Rouge:4, Vert:5, Jaune:5
                  printf("5: a = %d, b = %d, *p = %d.\n", a, b, *p);
  int * q = &b;   printf("6: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  *q *= (*p)++;   printf("7: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  *q -= ++(*p);   printf("8: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  p = q;          printf("9: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  q = &a;         printf("10:a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  return 0;
}
```

Bleu :

```
1: a = 3, b = 3, *p = 3.
2: a = 7, b = 3, *p = 7.
3: a = 7, b = 21, *p = 7.
4: a = 2, b = 21, *p = 2.
5: a = 2, b = 7, *p = 2.
6: a = 2, b = 7, *p = 2, *q = 7.
7: a = 3, b = 14, *p = 3, *q = 14.
8: a = 4, b = 10, *p = 4, *q = 10.
9: a = 4, b = 10, *p = 10, *q = 10.
10:a = 4, b = 10, *p = 10, *q = 4.
```

Rouge :

```
1: a = 4, b = 4, *p = 4.
2: a = 9, b = 4, *p = 9.
3: a = 9, b = 12, *p = 9.
4: a = 2, b = 12, *p = 2.
5: a = 2, b = 3, *p = 2.
6: a = 2, b = 3, *p = 2, *q = 3.
7: a = 3, b = 6, *p = 3, *q = 6.
8: a = 4, b = 2, *p = 4, *q = 2.
9: a = 4, b = 2, *p = 2, *q = 2.
10:a = 4, b = 2, *p = 2, *q = 4.
```

Vert :

```
1: a = 5, b = 5, *p = 5.
2: a = 8, b = 5, *p = 8.
3: a = 8, b = 15, *p = 8.
4: a = 4, b = 15, *p = 4.
5: a = 4, b = 3, *p = 4.
6: a = 4, b = 3, *p = 4, *q = 3.
7: a = 5, b = 12, *p = 5, *q = 12.
8: a = 6, b = 6, *p = 6, *q = 6.
9: a = 6, b = 6, *p = 6, *q = 6.
10:a = 6, b = 6, *p = 6, *q = 6.
```

Jaune :

```
1: a = 7, b = 7, *p = 7.
2: a = 11, b = 7, *p = 11.
3: a = 11, b = 21, *p = 11.
4: a = 6, b = 21, *p = 6.
5: a = 6, b = 4, *p = 6.
6: a = 6, b = 4, *p = 6, *q = 4.
7: a = 7, b = 24, *p = 7, *q = 24.
8: a = 8, b = 16, *p = 8, *q = 16.
9: a = 8, b = 16, *p = 16, *q = 16.
10:a = 8, b = 16, *p = 16, *q = 8.
```

Exercice 3 (4 points)

L'algorithme de `Babylone` calcule la racine carrée d'un nombre `A` avec une précision `P`. Il utilise une suite de nombres réels X_n tels que $X_{n+1} = (X_n + A/X_n)/2$. En C, programmer l'algorithme de `Babylone` en respectant les entrées sorties suivantes:

```
Calcul de la racine carree d'un nombre A avec une precision P.
Nombre A ? 2
Precision P ? 0.001
Valeur initiale ? 1.8
x1 = 1.455555, erreur = 0.344444
x2 = 1.414800, erreur = 0.040755
x3 = 1.414213, erreur = 0.000587
```

L'utilisateur entre le nombre `A`, la précision `P` et la valeur initiale X_0 de la suite. A chaque itération, le programme affiche la valeur de X_n et l'erreur $e = |X_n - X_{n-1}|$ avec 6 décimales. Le programme s'arrête lorsque l'erreur `e` est inférieure à `P`. On pourra utiliser des variables `a`, `p`, `x`, `xsave`, `e` et `n`. Dans cet exercice, on n'utilisera pas de fonction, ni de tableau. **(4 pts)**.

```
#include <stdio.h>

int main() {
  printf("Algorithme de Babylone.\n");
   printf("Calcul de la racine carree d'un nombre A avec une
precision P.\n");

  float x, a, precision;
  printf("Nombre A ? "); scanf("%f", &a);
  printf("Precision P ? "); scanf("%f", &precision);
  printf("Valeur initiale ? "); scanf("%f", &x);

  int i=1;
  float erreur;
  do {
    float xsave = x;
    x = (x + a/x)/2;
    printf("x%d = %.10f\n", i, x);
    erreur = x - xsave;
    if (erreur<0) erreur = -erreur;
    printf("erreur = %.10f\n", erreur);
    // printf("precision = %.10f\n", precision);
    i++;
  } while (erreur>precision);

  return 0;
}
```

Exercice 4 (8 points)

1) Ecrire une procédure `void affiche` prenant en entrée un tableau `t` d'éléments de type `float` et un nombre `n` d'éléments et affichant les `n` premiers éléments du tableau sans chiffre après la virgule et séparés par un espace.

```
// 1 point
void affiche(float * t, int l) {
  int i;
  for (i=0; i<l; i++)
    // printf("t[%d] = %.0f ", i, t[i]);
    printf("%.0f ", t[i]);
  printf("\n");
}
```

2) Ecrire une procédure `void init` prenant en entrée un tableau `t` d'éléments de type `float` et un nombre `n` d'éléments et initialisant le tableau avec des nombres aléatoires compris entre `0` et `9`.

```
// 1 point
void init(float * t, int l) {
  int i;
  for (i=0; i<l; i++) t[i]=rand()%MODULO;
}
```

3) Ecrire une procédure `void tamovar` prenant en entrée un tableau `t` d'éléments de type `float` et un nombre `n` d'éléments et donnant en sortie la moyenne `m` et la variance `v` des `n` éléments du tableau. Rappel mathématique:

$m = (\Sigma t[i])/n$ et $v = (\Sigma t[i]^2 - (\Sigma t[i])^2/n)/n$

```
// 3 points
void tamovar(float * t, int l, float * mo, float * var)
{
  int i; float s=0, s2=0;
  for (i=0; i<l; i++) {
    s += t[i];
    s2 += t[i]*t[i];
  }
  *mo = s/l;
  *var = (s2 - s*s/l)/l;
}
```

4) Ecrire un programme `main` déclarant un tableau de taille `10` initialisé avec `init`, affichant le tableau avec `affiche`, calculant la moyenne et la variance avec `tamovar`. et affichant les résultats avec deux chiffres après la virgule. Le programme respectera la sortie suivante.

```
3 6 7 5 3 5 6 2 9 1
moyenne = 4.70, variance = 5.41
```

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

#define TAILLE 10
#define MODULO 10

/// ici les fonctions des questions 1 2 3....

int main() {
  float tab[TAILLE];
  float m, v;
  // srand(time(NULL));
  srand(1);

  // 1 point
  init(tab, TAILLE);
  affiche(tab, TAILLE);

  // 1 point
  tamovar(tab, TAILLE, &m, &v);

  // 1 point
  printf("moyenne = %.2f, variance = %.2f\n", m, v);
  return 0;
}
```