ECUE «Introduction à la programmation »

Contrôle continu n°3 – 7 janvier 2015
sans document - durée 1 heure 30

CORRIGE CORRIGE CORRIGE

Exercice 1 (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)**.

```c
#include <stdio.h>

int main() {
  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 = %.6f\n", i, x);

    erreur = x - xsave;
    if (erreur<0) erreur = -erreur;
    printf("erreur = %.6f\n", erreur);

    i++;
  } while (erreur>precision);
  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]`. **(2pts)**.

```
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. **(2pts)**.

```
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 `3` 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. **(2pts)**.

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
```

```
#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 (10 points)

Soit `triRapide.c` le programme suivant.

```
#define TAILLE 8
void affiche(int * t, int premier, int dernier) {
  int i; printf("[ ");
  for (i=0; i<premier; i++) printf ("_ ");
  for (i=premier; i<=dernier; i++) printf ("%d ", t[i]);
  for (i=dernier+1; i<TAILLE; i++) printf ("_ ");
  printf("]\n");
}
int partition(int * t, int premier, int dernier) {
  int i, tmp, j = premier;
  for (i=premier; i<dernier; i++) {
    if (t[i] <= t[dernier]) {
      tmp = t[i]; t[i] = t[j]; t[j] = tmp; j++;
    }
    printf("P: i = %d j = %d ", i, j); affiche(t, premier, dernier);
  }
  tmp = t[dernier]; t[dernier] = t[j]; t[j] = tmp;
  printf("P: pivot = %d ", j); affiche(t, premier, dernier);
  return j;
}
void triRapide(int * t, int premier, int dernier, int niveau) {
  if (premier<dernier) {
    printf("TR debut: pre = %d der = %d niv = %d\n", premier, dernier, niveau);
    int pivot = partition(t, premier, dernier);
    triRapide(t, premier, pivot-1, niveau+1);
    triRapide(t, pivot+1, dernier, niveau+1);
    printf("TR fin: pre = %d der = %d niv = %d ", premier, dernier, niveau);
```

```
    affiche(t, premier, dernier);
  }
}
int main() {
  int tab[] = { 1, 3, 5, 0, 7, 6, 4, 2 };
  triRapide(tab, 0, TAILLE-1, 0);
  return 0;
}
```

1) Avec quelles valeurs de `premier`, `dernier` et `niveau`, `triRapide` est-elle appelée par `main` ? **(0.5 pt)**

```
0, 7, 0
```

2) a) Donner la sortie du programme jusqu'à la fin de la première exécution de `partition`. **(2 pts)**

```
TR debut: pre = 0 dern = 7 niv = 0
P: i = 0 j = 1 [ 1 3 5 0 7 6 4 2 ]
P: i = 1 j = 1 [ 1 3 5 0 7 6 4 2 ]
P: i = 2 j = 1 [ 1 3 5 0 7 6 4 2 ]
P: i = 3 j = 2 [ 1 0 5 3 7 6 4 2 ]
P: i = 4 j = 2 [ 1 0 5 3 7 6 4 2 ]
P: i = 5 j = 2 [ 1 0 5 3 7 6 4 2 ]
P: i = 6 j = 2 [ 1 0 5 3 7 6 4 2 ]
P: pivot = 2 [ 1 0 2 3 7 6 4 5 ]
```

2) b) En quelle position du tableau la dernière valeur du tableau a-t-elle été déplacée ? **(0.5 pt)**

```
En position 2
```

2) c) Cette position s'appelle le pivot. Quelle est la valeur du tableau au pivot ? **(0.5 pt)**

```
Sa valeur est 2.
```

2) d) Que vérifient les valeurs dont la position est inférieure à la valeur du pivot ? **(0.25 pt)**

```
Elles sont inférieures à celle du pivot.
```

2) e) Que vérifient les valeurs dont la position est supérieure à la valeur du pivot ? **(0.25 pt)**

```
Elles sont supérieures à celle du pivot.
```

3) a) Avec quelles valeurs de `premier` et `dernier`, `triRapide` de niveau 1 est-elle appelée la première fois par `triRapide` de niveau 0 ? la seconde fois ? **(1 pt)**

```
0 et 1 la première fois
3 et 7 la seconde fois
```

3) b) Donner la suite de la sortie du programme jusqu'à la fin de l'exécution du `main`. **(3 pts)**

```
TR debut: pre = 0 dern = 1 niv = 1
P: i = 0 j = 0 [ 1 0 _ _ _ _ _ _ ]
P: pivot = 0 [ 0 1 _ _ _ _ _ _ ]
TR fin: pre = 0 der = 1 niv = 1 [ 0 1 _ _ _ _ _ _ ]
TR debut: pre = 3 dern = 7 niv = 1
P: i = 3 j = 4 [ _ _ _ 3 7 6 4 5 ]
P: i = 4 j = 4 [ _ _ _ 3 7 6 4 5 ]
P: i = 5 j = 4 [ _ _ _ 3 7 6 4 5 ]
P: i = 6 j = 5 [ _ _ _ 3 4 6 7 5 ]
P: pivot = 5 [ _ _ _ 3 4 5 7 6 ]
TR debut: pre = 3 dern = 4 niv = 2
P: i = 3 j = 4 [ _ _ _ 3 4 _ _ _ ]
P: pivot = 4 [ _ _ _ 3 4 _ _ _ ]
TR fin: pre = 3 der = 4 niv = 2 [ _ _ _ 3 4 _ _ _ ]
TR debut: pre = 6 dern = 7 niv = 2
P: i = 6 j = 6 [ _ _ _ _ _ _ 7 6 ]
P: pivot = 6 [ _ _ _ _ _ _ 6 7 ]
TR fin: pre = 6 der = 7 niv = 2 [ _ _ _ _ _ _ 6 7 ]
TR fin: pre = 3 der = 7 niv = 1 [ _ _ _ 3 4 5 6 7 ]
TR fin: pre = 0 der = 7 niv = 0 [ 0 1 2 3 4 5 6 7 ]
```

3) c) Dessiner l'arbre d'appel des fonctions `triRapide` et `partition`. Chaque noeud de l'arbre correspondra à un appel de `triRapide` pour lequel `premier<dernier` est vrai ou à un appel de `partition`. Dans chaque noeud, on précisera les valeurs de `premier`, `dernier`. **(2 pts)**