Auteur : Mohamed Messabihi
Matière : Programmation et structures de données
Date 15 Mars 2017
Durée 1h30

Université Abou Bakr Belkaïd - Tlemcen
Faculté des Sciences
1ère Année MI
Semestre 2

Correction du Contrôle Continu

Aucun document n'est autorisé
Les solutions doivent être rédigées en **C**
Les appareils portables doivent être éteints et posés sur le bureau du surveillant

1 Affichage — 7 pts. 25'

1. Qu'affichent les deux programmes suivants :

```c
#include <stdio.h>
void Toto(int * A, int n)
{
    int a, *D=A, *F = &A[n-1];
    while(D < F)
    {
        a = *F;
        *F = *D;
        *D = a;
        D++;
        F--;
    }
}
int main()
{
    int i, A[10]={0,1,2,3,4,5,6};
    Toto(A, 6);
    for(i=0;i<6;i++)
        printf("%d\n", *(A+i));
}
```

```c
#include <stdio.h>
void Loulou(char *S,char c,int *P,int *n)
{
    int i=0;
    *n=0;
    while(S[i]!='\0')
    {
        if(S[i] == c)
            *(P + (*n)++) = i;
        i++;
    }
}
int main(){
    char S[100]="Toto&Loulou";
    char c='o';
    int i, n, T[10];
    Loulou(S,c,T,&n);
    for (i = 0; i <n; i++)
        printf("%d \n", *(T + i));
}
```

2. Que font les deux fonctions Toto et Loulou ? (deux lignes au maximum pour chacune d'entre elles.)

Solution

1.

Affichage

```
5
4
3
2
1
0
```

Affichage

```
1
3
6
9
```

2. La fonction Toto **inverse l'ordre** des éléments du tableau T passé comme paramètre, et la fonction Loulou **construit un tableau** qui contient les **positions de toutes les occurrences** du **caractère 'o'** dans la chaine S.

2 Union et intersection de deux tableaux — 8 pts. 35'

On dispose de deux tableaux T et S d'entier à une seule dimension. Les tableaux T et S sont supposés déjà triés dans l'ordre croissant ↗ et sans doublons. La taille du tableau T est lt et celle du tableau S est ls. Les variables lt et ls doivent être inférieures ou égales à la taille maximale des tableaux T et S (fixée ici à 100).

1. Écrire une fonction Tab_Union qui prend en entrée deux tableaux T et S ainsi que leurs tailles réelles puis elle construit un tableau TuS de taille lTuS qui contient l'**union** des deux tableaux T et S. Le tableau TuS doit rester trié par construction.
2. Écrire une fonction Tab_Intersection qui prend en entrée deux tableaux T et S ainsi que leurs tailles réelles lt et ls puis elle construit un tableau TnS de taille lTnS qui contient l'**intersection** des deux tableaux T et S. Le tableau TnS doit rester trié par construction.

Exemple :

T :

3	8	11	17	23	48	56	61	87	93	98

lt = 11.

S :

4	6	8	23	53	56	76	87	90

ls = 9.

TuS :

3	4	6	8	11	17	23	48	53	56	61	76	87	90	93	98

lTuS = 16.

TnS :

8	23	56	87

lTnS = 4.

Solution

1.

```
1 void Tab_Union(int T[100], int lt, int S[100], int ls, int TuS[100], int *lr)
2 {
3     int i,j,k;
4     i=0; j=0; k=0; *lr=0;
5     while(i<lt && j<ls)
6         if (T[i]==S[j])
7             { TuS[k++]=T[i]; i++; j++;}
8         else if (T[i]<S[j])
9             {TuS[k++]=T[i++]; }
10            else
11                TuS[k++]=S[j++];
12    while(j<ls)
13            TuS[k++]=S[j++];
14    while(i<lt)
15            TuS[k++]=T[i++];
16    *lr=k;
17 }
```

2.

```
1 void Tab_Intersection(int T[100],int lt,int S[100],int ls,int TnS[100],int *lr)
2 {
3     int i,j,k;
4     i=0; j=0; k=0; *lr=0;
5     while(i<lt && j<ls)
6         if (T[i]==S[j])
7             { TnS[k++]=T[i]; i++; j++;}
8         else if (T[i]<S[j])
9                 i++;
10            else
11                j++;
12    *lr=k;
13 }
```

3 Le tableau génératif — 5 pts. 30'

Écrire une fonction Tab_Generatif qui prend en entrée un tableau T (à deux dimensions) et un nombre d'étapes N, qui le rempli avec le chiffre 1, 2 et 3 en appliquant le principe suivant :

1. Au début, à l'étape 1 : la première case de la première ligne contient le chiffre 1.

2. Puis à chaque étape de la génération (*c.-à-d,* la ligne suivante), 1 est transformé en 1 2 3 (3 cases) , 2 est transformé en 3, et 3 est transformé en 1.
3. Toutes les autres cases sont initialisées à 0.

Exemple :

```
Etape 1 : 1
Etape 2 : 1 2 3
Etape 3 : 1 2 3 3 1
Etape 4 : 1 2 3 3 1 1 1 2 3
Etape 5 : 1 2 3 3 1 1 1 2 3 1 2 3 1 2 3 3 1
Etape 6 : 1 2 3 3 1 1 1 2 3 1 2 3 1 2 3 3 1 1 2 3 3 1 1 2 3 3 1 1 1 2 3
Etape 7 : ...
```

L'appelle de la fonction Tab_Generatif avec un nombre d'étapes égale à 6 nous permet donc d'obtenir le tableau ci-dessous :

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	3	1	1	1	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	3	1	1	1	2	3	1	2	3	1	2	3	3	1	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	3	1	1	1	2	3	1	2	3	1	2	3	3	1	1	2	3	3	1	1	2	3	3	1	1	1	2	3

Solution

```c
void Tab_Generatif(int T[100][100], int etapes, int *col)
{
    int n1=1,n2=1;
    int i, j1,j2;
    for(j1=0; j1<100; j1++)
        for(j2=0; j2<100; j2++)
            T[j1][j2]=0;
    T[0][0]=1;
    for(i=1 ; i<etapes; i++)
     {
         j1=0; j2=0;
         while(j1<n1)
         {
            if (T[i-1][j1]==2) {T[i][j2]=3 ; j1++; j2++;}
            else if (T[i-1][j1]==3) {T[i][j2]=1 ; j1++ ;j2++;}
            else
            {
                T[i][j2++]=1;
                T[i][j2++]=2;
                T[i][j2++]=3;
                n2=n2+2;
                j1++;
            }
          }
          n1=n2;
     }
     *col=n2;
}
```

☞**Remarque.** *Pour les deux derniers exercices :*

1. *On ne demande ni la saisie des tableaux ni leur affichage.*
2. *La manipulation des tableaux avec les crochets* [] *ou* [][] *est beaucoup plus simple et pratique que la notation avec pointeurs.*
3. *On ne demande pas l'écriture de la fonction* main *pour tester les autres fonctions.*

« Bon courage »