TP 6 – Allocation dynamique de mémoire et dans une deuxième partie : boîte à outils

1 Objectifs du TP

Ce TP a pour but de découvrir comment :

1. allouer dynamiquement de la mémoire ;
2. utiliser des points de reprises ;
3. utiliser les fonctions d'horloge ;
4. utiliser le générateur de nombres aléatoires,

2 Allocation statique et dynamique de mémoire

Nous nous intéresserons plus particulièrement aux tableaux ; mais tout est transposable aux variables simples ou aux variables structurées. Afin de pouvoir paramétrer modifier facilement les programmes que nous écrirons nous utiliserons des facilités du langage C à savoir :

1. La définition de constantes symboliques :
   #define CONSTANTE valeur
2. La définition de types synonymes :
   typedef <ancien_nom> <nouveau_nom> ;
   par exemple : typedef int entier ;

Créer un nouveau répertoire pour y mettre les fichiers de ce TP. Dans ce répertoire, créer un fichier *tableau.c* contenant le code suivant :

```
1.       #include <stdio.h>
2.       #define QTE 10
3.
4.       typedef char MON_TYPE;
5.
6.       int main (void)
7.       {
8.       MON_TYPE tablo[QTE] ;
9.             printf("Taille du type initial : %d \n",sizeof(MON_TYPE));
10.            printf("Taille de la variable tablo : %d\n",sizeof(tablo));
11.            printf("Adresse de la variable tablo : %p\n",&tablo); /* Le
               compilateur laisse passer a priori cette erreur !!! */
12.            printf("Adresse du premier élément : %p\n",&tablo[0]);
13.            printf("Taille d'un élément de tablo : %d\n",sizeof(tablo[0]));
14.            return (EXIT_SUCCESS) ;
15.      }
```

Q. 1 Exécuter le programme. Faire varier *QTE* et le *type synonyme* pour vérifier les règles de proportionnalité.

Nous allons maintenant effectuer une allocation dynamique de ce même tableau en utilisant un appel à la fonction **malloc** dont le prototype ainsi que celui de la fonction **exit** est défini dans <stdlib.h> (cf pages de man).

Créer un nouveau fichier *tableauDyn.c* reprenant le code précédent dans lequel on aura substitué la ligne 8 par les 3 lignes suivantes :

```
1.      MON_TYPE * tablo ;
2.      tablo = (MON_TYPE *) malloc (QTE * sizeof(MON_TYPE));
3.      if (tablo==NULL) {
          perror("Echec malloc") ;
          exit (EXIT_FAILURE) ;
          }
```

Q. 2 Exécuter le programme. Faire varier *QTE* et le *type synonyme* et vérifier les règles de proportionnalité. Que se passe-t-il ? Expliquer au vu du cours. Faire un dessin.

En fait pour connaître la taille de la zone allouée utilisable il existe un appel système : **malloc_usable_size** (cf pages de man).

Modifier dans votre programme précédent la ligne d'affichage de la taille du tableau.

Q. 3 Reprendre la question Q1 et faire des remarques.

Modifier maintenant votre programme en remplaçant l'appel de malloc par celui de calloc (cf pages de man).

Q. 4 Reprendre la question Q1.

On souhaite modifier, en cours d'exécution, la taille du tableau *tablo*. Afin de bien visualiser ce qui se passe nous utiliserons 3 variables dynamiques afin d'encadrer :

```
1.      #include <stdio.h>
2.      #include <stdlib.h>
3.      #include <malloc.h>
4.
5.      #define QTE 12
6.
7.      typedef char MON_TYPE;
8.
9.      int main (void)
10.     {
11.       MON_TYPE * t1 ;
12.       MON_TYPE * tablo ;
13.       MON_TYPE * t2 ;
14.       MON_TYPE * nouveau ;
15.
```

```
16.      t1= (MON_TYPE *) malloc (4 * sizeof(MON_TYPE));
17.      if (t1==NULL) { perror("Echec malloc") ; exit (EXIT_FAILURE) ;}
18.
19.      tablo= (MON_TYPE *) malloc (QTE * sizeof(MON_TYPE));
20.      if (tablo==NULL) { perror("Echec malloc") ; exit (EXIT_FAILURE) ;}
21.
22.      t2= (MON_TYPE *) malloc (4 * sizeof(MON_TYPE));
23.      if (t2==NULL) { perror("Echec malloc") ; exit (EXIT_FAILURE) ;}
24.
25.      printf("\nValeur de la variable t1 : %p\n",t1);
26.      printf("Valeur de la variable tablo : %p\n",tablo);
27.      printf("Valeur de la variable t2 : %p\n\n",t2);
28.
29.      nouveau= (MON_TYPE *) realloc (tablo,QTE/10 * sizeof(MON_TYPE));
30.      if (nouveau==NULL) { perror("Echec realloc") ; exit (EXIT_FAILURE) ; }
31.      if (nouveau==tablo) { printf("Même emplacement \n") ; }
32.          else  { printf("Nouvel emplacement : %p\n",nouveau) ; }
33.      printf("Espace utilisable variable tablo : %d\n",
               malloc_usable_size(nouveau));
34.
35.      nouveau= (MON_TYPE *) realloc (tablo,QTE*10 * sizeof(MON_TYPE));
36.      if (nouveau==NULL) { perror("Echec realloc") ; exit (EXIT_FAILURE) ; }
37.      if (nouveau==tablo) { printf("Même emplacement \n") ; }
38.          else  { printf("Nouvel emplacement %p\n",nouveau) ; }
39.      printf("Espace utilisable variable tablo : %d\n",
         malloc_usable_size(nouveau));
40.      free(t1) ;
41.      free(t2) ;
42.      free(nouveau) ;
43.      }
```

⚠ Créer un nouveau fichier *ReallocDyn.c* reprenant le code ci-dessus.

Q. 5 Après avoir commenté ce programme, reprendre la question Q1. Faire un dessin du tas pour QTE = 15 et MON_TYPE ↔ int .

Q. 6 Exercice de synthèse

Considérons la structure ChaineDyn suivante qui contient l'adresse d'un tableau de caractères et son sombre d'éléments.

```
1.      typedef struct
2.      {
3.        long int nbCar ;
4.        char *tab ;
5.      } ChaineDyn ;
```

1 Écrire une fonction qui crée une ChaineDyn d'au plus n caractères dont l'entête est :

```
1.      Chainedyn creerChaine(const int n)
```

2 Écrire une fonction qui détruit une ChaineDyn dont l'entête est :

```
1.      void detruireChaine(ChaineDyn * ch) ;
```

3 Écrire une fonction qui lit une ChaineDyn dont l'entête est :

```
1.      void lireChaine(ChaineDyn * ch) ;
```

4 Écrire une fonction qui affiche une ChaineDyn dont l'entête est :

```
1.      void afficheChaine(const ChaineDyn ch) ;
```

5 Écrire un programme principal qui demande la taille d'une chaîne, la crée, indique son adresse et sa taille réelle, la lit puis l'affiche avant de la détruire.

6 Même question (la Q6.5) avec 3 ChainesDyn statiques qui sont définies par :

```
1.      ChaineDyn troisCh[3] ;
```

7 Après avoir fait un dessin, même question (la Q6.5) avec 3 ChaineDyn dynamiques qui sont définies par :

```
1.      ChaineDyn * troisCh[3] ;
```

8 (bonus) Adaptez la question 7 pour que le nombre de chaînes soit dynamique. Inspirez-vous de la question Q2

3 Générateur de nombre aléatoires

Dans cette partie nous allons nous poser la question de la génération d'une séqunce de nombres aléatoires. Soit le programme :

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  int alea(void)
5.  { return(rand()) ;}
6.
```

```
7.  int main (void)
8.  {
9.    printf("hasard %d \n", alea()) ;
10. }
11.
```

Q. 7 Analyser et modifier :

1 Tapez et exécutez plusieurs fois ce programme. Que constatez-vous ?

2 Appelez dans une boucle 5 fois la fonction alea(). Exécutez plusieurs fois votre programme. Que constatez-vous ?

3 Lire la page de man de la fonction clock() et ajoutez un appel à cette fonction en début de votre porgramme. Affichez la valeur renvoyée. Que constatez-vous ?

4 Lire les page de man en section 3 de rand() et celle de srand(). A partir des éléments que vous connaissez maintenant, modifiez le programme précédent pour que la séquence de nombre soit différente (a priori) à chaque exécution.

5 Modifiez votre programme pour renvoyer une valeur entre 0 et 1.

4 Les points de reprise pour simuler les exceptions

⚠ La compréhension du cours sur les points de reprise est fondamentale pour la suite.

Soit une fonction f(x) définie si et seulement si x est différent de zéro. Par exemple f(x) = 1/x.

Q. 8 Écrire cette fonction f(x) ainsi qu'un programme qui demande à l'utilisateur de taper une valeur réelle puis qui affiche soit f(x) soit un message d'erreur parce que la valeur saisie était nulle.

Cette première approche s'éloigne un peu du principe de prérequis et un exemple de mise en œuvre utilisant les points de reprises pourrait être le suivant :

```
2.      #include <stdio.h>
3.      #include <setjmp.h>
4.      #include <stdlib.h>
5.
6.      // Pour écrire un programme propre on définit un type exception
7.      typedef enum {OK, DIVISION_PAR_ZERO}  Exception ;
8.
9.      double f (const double x,  jmp_buf ptRep)
10.     {
11.       // saut au point de reprise si un prérequis n'est pas respecté
12.       if  (x == 0) {
13.           longjmp( ptRep, DIVISION_PAR_ZERO);
14.           } ;
15.       return ( 1 / x) ;
16.     }
17.
18.     void lireDouble(double *val)
```

```
19.         {
20.           char tampon[2048] ;
21.           char *err ;
22.           fgets(tampon,2048,stdin);
23.           *val=strtod(tampon,&err);
24.           if ((*err!='\0') && (*err!='\n')) { perror ("\n Attention
    caractère invalide dans le nombre fourni\n");}
25.         }
26.
27.       int main(void){
28.         jmp_buf ptRep;
29.         Exception anomalie;
30.         double valeur,resultat ;
31.
32.         //pose du point de reprise
33.             anomalie=setjmp(ptRep) ;
34.
35.         // Test de la valeur de retour
36.         switch (anomalie) {
37.           case OK :
38.                     printf("Entrer un réel : \n");
39.                     lireDouble(&valeur);
40.                     printf("f(%lf)=%lf\n",valeur,f(valeur,ptRep));
41.                     break;
42.           case DIVISION_PAR_ZERO :
43.                     printf("\n Opération impossible : division par zéro \n")
    ;
44.                     break;
45.           default :
46.                     printf("\n Anomalie inconnue  \n") ;
47.           }
48.         return (anomalie);
49.       }
```

Q. 9 Tapez ce programme et exécutez le pour différentes valeurs

Remplacer la fonction f(x) = 1/x par f(x)=1/sqrt(x) (cf bibliothèque standard ou pages de man). Dans ce cas là il y a 2 prérequis x /= 0 pour le calcul de 1/x et x ≥ 0 pour le calcul de sqrt(x).

⚠ Il faudra compiler votre nouveau programme en utilisant l'option -lm afin d'effectuer l'édition de liens avec la bibliothèque mathématique.

Q. 10 Modifiez proprement votre programme pour prendre en compte cette nouvelle spécification.

Q. 11 Sans utiliser de boucle explicite, modifier simplement votre programme pour que l'utilisateur soit obligé de fournir une valeur valide.