MONNERET Nicolas
ESTEVE Mathieu

Rapport de projet LO21

Tri à sceaux

Le but de ce projet, donné dans le cadre de l'UV LO21 est de construire un programme de tri à sceaux en langage C. L'avantage du tri à sceaux est de pouvoir trier des nombres dans différentes bases (ici de 2 à 16). Le principe consiste à diviser les nombres en chiffres et de trier dans l'ordre croissant les chiffres se situant au même niveau dans les nombres. On commence par comparer les derniers chiffres des nombres et on les trie suivant ces chiffres. Ensuite on remonte jusqu'aux premiers chiffres des nombres toujours en les triant à chaque fois. Le résultat de ces opérations est l'obtention des nombres triés par ordre croissant.

Ce rapport de projet décrit toutes les étapes de la conception du programme qui ont permis d'aboutir à une version finale en langage C qui fonctionne.

I) Préparation du projet

1) Premières idées

Nous nous sommes réunis et avons discuté du projet et des différentes façons de résoudre le problème. Le premier point sur lequel nous nous sommes mis d'accord fut de diviser les nombres en chiffres afin de pouvoir comparer les chiffres se situant à un même niveau dans les nombres. Chaque nombre sera stocké dans un sceau suivant la valeur du chiffre sur lequel on se trouve. Par exemple, si le dernier chiffre du nombre est 1, il sera stocké dans le sceau 1. Une fois tous les nombres que l'utilisateur a rentré stockés dans des sceaux, le programme va recomposé la liste de nombre en commençant par le sceau 0 jusqu'au dernier sceau. On va donc obtenir une liste de nombre ordonnée suivant leur dernier chiffre. Ensuite, on recommence l'opération avec le chiffre précédant dans le nombre et ceci jusqu'au premier chiffre.

2) Choix des représentations des données en C

Comme l'indique l'énoncé du projet, chaque chiffre doit être représenté par un caractère car il s'agit d'un chiffre dans une base quelconque. Nous choisissons donc de représenter les nombres non pas comme des entiers où des réels mais comme des chaînes de caractère. Cela permet d'accéder facilement à chaque chiffre de la chaîne de caractères dans le but de les comparer.

La liste des nombres que l'utilisateur va entrer afin de les trier doit être représenté par une structure de données dynamique car le nombre de nombres va changer à chaque utilisation du programme. Nous avons donc choisi d'utiliser une liste chaînée. La suite de nombres sera donc une liste chaînée de chaîne de caractères.

Les sceaux doivent également être représentés de façon dynamique car leur taille va varier entre chaque étape du processus. Nous décidons donc de les représenter également par des listes chaînées de chaînes de caractères.

Ces sceaux seront contenus dans un tableau à une dimension. La taille du tableau va être égale à la base des nombres. Chaque case du tableau contiendra un sceau, le sceau 0 dans la case d'indice 0…

On aura donc un tableau de listes chaînées de chaînes de caractères.

3) Réalisation de l'algorithme

Après toutes ces mises au point nous avons réalisé un algorithme simple de départ.

a) Programme principal

Le programme principal utilise une fonction qui effectue le tri des nombres suivant le chiffre désigné. On incrémente la position du chiffre désigné avec une boucle « pour » et on effectue le tri pour chaque chiffre désigné, ceci jusqu'à ce que le nombre ai été trié à l'aide de tous ces chiffres. L'algorithme est le suivant :

```
Pour i de (maxchiffres-1) à 0 par pas de -1 faire
    tri (l, i, base)
fin pour
```

b) La fonction tri

La fonction tri va trier les nombres suivant le chiffre désigné par le programme principal. L'algorithme est le suivant :

```
Tant que la liste n'est pas vide faire
    Choix (((valeur (l)) [i])
        Cas 0 : sceaux [0] = ajoutQ (sceaux [0], valeur (l))
        Cas 1 : sceaux [1] = ajoutQ (sceaux [1], valeur (l))
        Cas 2 : sceaux [2] = ajoutQ (sceaux [2], valeur (l))
        Cas 3 : sceaux [3] = ajoutQ (sceaux [3], valeur (l))
        Cas 4 : sceaux [4] = ajoutQ (sceaux [4], valeur (l))
        Cas 5 : sceaux [5] = ajoutQ (sceaux [5], valeur (l))
        Cas 6 : sceaux [6] = ajoutQ (sceaux [6], valeur (l))
        Cas 7 : sceaux [7] = ajoutQ (sceaux [7], valeur (l))
        Cas 8 : sceaux [8] = ajoutQ (sceaux [8], valeur (l))
        Cas 9 : sceaux [9] = ajoutQ (sceaux [9], valeur (l))
        Cas A : sceaux [10] = ajoutQ (sceaux [10], valeur (l))
        Cas B : sceaux [11] = ajoutQ (sceaux [11], valeur (l))
        Cas C : sceaux [12] = ajoutQ (sceaux [12], valeur (l))
        Cas D : sceaux [13] = ajoutQ (sceaux [13], valeur (l))
        Cas E : sceaux [14] = ajoutQ (sceaux [14], valeur (l))
        Cas F : sceaux [15] = ajoutQ (sceaux [15], valeur (l))
    l = suivant (l)
Fin tant que

Pour j de 0 à base par pas de 1 faire
    Tant que sceaux [j] n'est pas vide faire
        l = ajoutQ (l, valeur (sceaux[j])
        sceaux [j] = suivant (sceaux[j])
    fin tant que
fin pour
retourner l
```

(Valeur (l)) [i] est le nombre à entrer dans un sceau, i est la place du chiffre désignée par le programme principal avec lequel on va effectuer le tri. Pour chaque cas (valeur du chiffre), on va rentrer le nombre dans le sceau correspondant en l'ajoutant en queue de la liste déjà présente dans ce sceau. Cet ajout en queue va permettre de conserver les tris effectués lors des étapes précédentes.

Ensuite, le programme va reprendre chaque sceau du premier au dernier et ajouter en queue de la liste de nombres le contenu du sceau. Cet ajout en queue va également conserver les tris précédents.

Cette fonction étant reprise dans le programme principal pour chaque chiffre des nombres, à la fin du programme, on obtient une liste de nombres triés.

II) Traduction en C et ajout de fonctions

Après avoir mis au point un algorithme simple qui résout notre problème, il a fallut passer à l'étape suivante : la mise au point en langage C et l'ajout de fonctions pour l'acquisition des données, le contrôle des entrées, etc.

1) Modifications pour le langage C

a) Définition des types

Avant toute choses, il faut définir le type liste chaînée, pour cela on définit au préalable le type element :

```
typedef struct elem {
        char valeur [11] ;
        struct elem * suivant;
        } element;

Typedef element * liste;
```

b) Modifications de la fonction tri et du programme principal

On modifie tout d'abord le programme principal en allouant un espace à la liste l par la fonction malloc. Ensuite on créé une fonction acquisition qui va permettre de demander à l'utilisateur les variables utilisées et une fonction affichage pour afficher le résultat.

```
l = (element*) malloc (sizeof (element))
l = acquisition (&base, &maxchiffres)

pour i de maxchiffres-1 à 0 par pas de -1
      l = tri(l,i,base)
fin pour

affichage (l)
```

Dans la fonction tri, il faut également allouer des places mémoire à chaque liste du tableau sceaux. On fait ceci pour chaque case du tableau à l'aide d'une boucle « pour »:

```
Pour j de 0 à base par pas de 1
        sceaux [j] = (element*)malloc (sizeof (element))
        sceaux [j] = vide
Fin Pour
```

A la fin de la fonction tri, il ne faut pas oublier de restituer la mémoire à l'aide de la fonction free :

```
……
        temp = l
        l = suivant (l)
        free (temp)
fin tant que
pour j de 0 à base par pas de 1
        tant que sceaux [j] n'est pas vide
                l = ajoutQ (l, valeur(sceaux[j]))
                temp = sceaux [j]
                sceaux [j] = suivant (sceaux [j])
                free (temp)
        fin tant que
        free (sceaux [j])
fin pour
retourner l
```

2) Les fonctions principales

a) la fonction ajoutQ

Cette fonction sert à ajouter un élément en queue d'une liste chaînée. L'algorithme :

```
Données : liste l, chaîne de caractères e
liste nouv
liste temp

suivant (nouv) = vide
strcpy (valeur (nouv),e)

temp = l
si l est vide alors l = nouv
sinon   tant que suivant (temp) n'est pas vide
                temp = suivant (temp)
        fin tant que
        suivant (temp) = nouv
fin si
retourner l
```

On utilise la fonction strcpy () afin de copier une chaîne de caractère sur une autre car le signe = ne fonctionne pas sur les chaînes de caractère.

b) La fonction affichage

Cette fonction permet d'afficher à l'utilisateur la liste chaînée finale qui contient les nombres triés.

```
Données : liste l
liste temp
entiers i,j

temp = l
écrire (Voici votre liste)
tant que temp n'est pas vide
        i = 0
        tant que (valeur (temp)[i] = '0') et (i < strlen (valeur (temp)-1))
                i = i+1
        fin tant que
        pour j de i à strlen (valeur (temp)-1) par pas de 1
                écrire (valeur (temp)[j])
        fin pour
        temp = suivant (temp)
fin tant que
```

La fonction traite les nombres un par un en affichant chaque chiffre des nombres un par un également. On se retrouve ainsi avec deux boucles imbriquées : une pour les nombres et une pour les chiffres de chaque nombre.

Ici, on n'affiche pas les 0 contenus avant le nombre en utilisant une boucle tant que. Par exemple si le nombre 45 est traité dans le programme comme 0045, la fonction affichage ne va afficher que 45 et pas 0045.

La fonction strlen permet d'obtenir la longueur de la chaîne de caractère contenue dans l'élément de la liste temp.

3) Les fonctions d'acquisition

a) la fonction max

Cette fonction permet de trouver le nombre de chiffre que possède le plus long des nombres entrés par l'utilisateur. L'algorithme est :

```
Données : liste l
entier x = 0
liste p

p = l
tant que p n'est pas vide
        si (strlen (valeur (p)) > x)
                x = strlen (valeur (p))
        fin si
```

```
        p = suivant(p)
fin tant que
retourner x
```

Ici, on compte le nombre de chiffre à l'aide de la fonction strlen qui renvoie la taille de la chaîne de caractères. A l'aide du « si », on ne garde que la longueur du nombre le plus long. On effectue ceci sur tous les éléments de la liste de nombres.

b) La fonction modif

Cette fonction a pour but de modifier la liste l afin que tous les nombres à l'intérieur entrés par l'utilisateur aient la même taille. Ceci afin que la fonction tri puisse les placer dans les sceaux et les trier correctement. Cette fonction modif rajoute donc des 0 au début des nombres qui sont de taille inférieure à la taille du plus grand nombre de la liste l.

```
Données : liste l, entier x
entier i
chaîne de 30 caractère temp
liste p
p = l
tant que p n'est pas vide
        strcpy (temp,"\0")
        si (strlen (valeur (p)) < x)
                pour i de 1 à (x-strlen (valeur (p))) par pas de 1
                        strcat (temp,"0")
                fin pour
                strcat (temp, valeur(p))
                strcpy (valeur(p),temp)
        fin si
        p = suivant(p)
 fin tant que
 retourner l
```

La fonction strlen sert à calculer la longueur d'une chaîne de caractères. La fonction strcpy sert à copier une chaîne de caractère dans une autre. La fonction strcat sert à concaténer deux chaînes de caractères.

c) la fonction acquisition

La fonction acquisition nous permet de définir les nombres qui seront contenus dans la liste mais aussi de choisir dans quelle base nous voulons travailler. On entre en paramètres deux adresses, celles des variables base et maxchiffres pour les définir dans cette fonction.

Cette fonction demande d'abord à l'utilisateur de choisir une base entre 2 et 16 pour les nombres avec lesquelles il veut travailler avec un contrôle pour vérifier que la valeur qu'il entre corresponde aux exigences du programme. Cette valeur est stockée à l'adresse de base.

Le programme va ensuite demander à l'utilisateur de taper un nombre dans cette base. Bien entendu, le programme va d'abord vérifier que celui-ci est bien compris dans la base choisie précédemment. Pour cela, on va créer une boucle Répeter…Tant que. Dans cette boucle, on va utiliser une deuxième boucle Tant que imbriquée ainsi qu'une variable bon qu'on initialise à vrai (1 en C) avant cette seconde boucle et un compteur i.

Dans cette boucle Tant Que qui continue tant que i<strlen(nb) et bon=vrai, on effectue un contrôle sur le code ASCII du chiffre nb[i]. On a deux cas possibles de test :

- le premier cas est si la base est inférieure ou égale à 10. Dans ce cas le contrôle est:

```
Si  (nb[i] < 48 ou nb[i] > (47 + base))
        Alors bon=faux
        Sinon rien
Fin Si
```

- le second s'effectue dans l'autre cas, c'est-à-dire si la base est supérieure à 10. Dans ce cas, le contrôle est :

```
Si ( nb[i] < 48 OU nb[i] > 65+base-11) OU ( nb[i] > 58 ET nb[i] < 65)
```

Le problème vient du fait que le code ASCII de A ne se trouve pas juste après celui de 9. En effet le code ASCII de 9 est 58 et celui de A est 65. Donc lorsque la base est supérieur à 10, il faut contrôler que le code ASCII du chiffre que l'on est an train de vérifier appartienne à un de ces deux intervalles.

Si le contrôle s'avère être faux, la valeur bon passe à Faux et la boucle Tant Que se termine. On arrive donc au test de notre boucle Répéter…Tant Que qui teste la valeur bon est qui recommence si la valeur bon est fausse.

Après ce petit contrôle, on ajoute en queue de la liste l la chaine de caractère et on demande à l'utilisateur s'il veut rentrer un autre nombre. Pour cela, on utilise une boucle Répéter…Tant Que qui englobe toute la fonction principale.

L'algorithme de la fonction est donc le suivant :

```
liste acquisition( entier base, entier maxchiffres)

entier i
booléen bon
liste l
chaîne de caractères nb[11] ,  rep
écrire ("              TRI A SCEAUX")
écrire ("\n\nVous allez entrer une liste de nombre afin de les trier")
Répéter
        ecrire(Dans quelle base travaillez-vous ? ( entre 2 et 16))
        lire(base)
Tant que ( base<2 OU base>16)
l=vide
Répéter
        Répéter Tant Que
                bon=vrai
                écrire (Taper un nombre dans la base base)
                lire nn
                i=0
                Tant Que ( i<strlen(nb) ET bon=1 )
                        Si base <10
                                Alors Si  (nb[i] < 48 ou nb[i] > (47 + base))
                                        Alors Bon = faux
```

```
                        Fin Si
                Sinon Si (nb[i]<48 OU nb[i]> 65+base-11) OU ( nb[i]>58 ET nb[i]<65)
                        Alors Bon = faux
                    Fin Si
                Fin Si
                i=i+1
            Fin Tant Que
        Tant Que (bon=0)
         l= ajoutQ ( l , nb )
        Répéter
            Ecrire (Voulez-vous entrer un autre nombre (O/N))
            Lire ( rep )
        Tant Que ( rep!='O' ET rep!='N' )
    Tant Que ( rep='O' )
    maxchiffres=max(l)
    l = modif ( l , maxchiffres )
    retourner l
```

Conclusion

Nous venons de vous décrire dans ce rapport la manière dont nous avons conçu et pensé ce programme de manière chronologique. Nous avons expliqué le but et l'intérêt du programme ainsi que son fonctionnement. Chaque fonction a également été décrite dans ce rapport : fonctionnement et but. Nous avons aussi décrit la manière dont les données entrées par l'utilisateur sont traitées (chaînes de caractères, tableau de listes etc.

Vous pourrez trouver en annexes toutes les lignes de code du programme ainsi que chaque fonction et définition de type en langage C.

Annexes : code en C

```
#include <stdio.h>
#include <stdlib.h>

typedef struct elem {
        char valeur[11];
        struct elem* suivant;
        }element;

typedef element* liste;

void affichage (liste l)
{
        liste temp;
        int i,j;

        temp=l;
        printf("\nVoici votre liste :\n");
        while(temp!=NULL)
        {
                i=0;
      while((temp->valeur[i]=='0')&&(i<strlen(temp->valeur)-1))
      {
        i=i+1;
      }
      for(j=i;j<=strlen(temp->valeur)-1;j++)
      {
        printf("%c",temp->valeur[j]);
      }
      printf("\n");
                temp=temp->suivant;
        }
}

liste ajoutQ (liste l, char e[])
{
        liste nouv;
        liste temp;

        nouv=(element*)malloc(sizeof(element));
        nouv->suivant=NULL;
   strcpy(nouv->valeur,e);

        temp=l;
        if(l==NULL)
        {
                l=nouv;
        }
```

```
	else
	{
		while(temp->suivant!=NULL)
		{
			temp=temp->suivant;
		}
		temp->suivant=nouv;
	}
	return l;
}


int max(liste l)/*trouve le nombre de chifrres que possède le plus long des nombres*/
{
 int x=0;
 liste p;
 p=l;
 while(p!=NULL)
 {
    if(strlen(p->valeur)>x)
    {
       x=strlen(p->valeur);
    }
    p=p->suivant;
 }
 return x;
}


liste modif(liste l, int x)/*modifie la liste l pour que tous les nombres à l'interieur de l aient exactement le même nombre de chiffres en rajoutant des zéros à certain*/
{
int i;
char temp[30];
liste p;
p=l;
while(p!=NULL)
{
	strcpy(temp,"\0");
	if(strlen(p->valeur)<x)
	{
		for(i=1;i<=(x-strlen(p->valeur));i++)
		{
			strcat(temp,"0");
		}
		strcat(temp,p->valeur);
		strcpy(p->valeur,temp);
	}
	p=p->suivant;
}
```

```
return l;
}

liste acquisition(int *x, int *maxchiffres)
{
int i,bon;
liste l;
char nb[11],rep;

printf("              TRI A SCEAUX");
printf("\n\nVous allez entrer une liste de nombre afin de les trier");
do
{
        printf("\n\nDans quelle base voulez-vous travailler? \nVotre base doit etre
        comprise entre 2 et 16 : ");
        scanf("%d",&(*x));
}
while(((*x)<2)||((*x)>16));
l=(element*)malloc(sizeof(element));
l=NULL;
do
{
        do
        {
                bon=1;/*on initialise le booleen à vrai*/
                printf("\nTaper un nombre dans la base %d :",(*x));
                getchar();
                gets(nb);
                i=0;
                while((i<strlen(nb))&&(bon==1))
                {
                        if((*x<10))
                        {
                                if((nb[i]<48)||(nb[i]>(47+(*x))))/*cas où le caractère visé
                                n'est pas compris dans la base choisie*/
                                {
                                        bon=0;/*booleen devient faux*/
                                }
                        }
                        else
                        {
                                if(((nb[i]<48)||(nb[i]>(65+(*x)-11)))||((nb[i]>58)&&(nb[i]<65)))
                                {
                                        bon=0;
                                }
                        }
                        i=i+1;
                }
        }
         while(bon==0);
```

```
	l=ajoutQ(l,nb);
	 do
	{
		printf("Voulez-vous entrer un autre nombre (O/N)");
		scanf("%c",&rep);
	}
	while((rep!='O')&&(rep!='N'));
}
while(rep=='O');
*maxchiffres=max(l);
l=modif(l,*maxchiffres);
return l;
}

liste tri (liste l, int i, int base)
{
	int j;
	liste sceaux[base];
	liste temp;
	char nombre[11];

	 for(j=0;j<base;j=j+1)
	{
		sceaux[j]=(element*)malloc(sizeof(element));
		sceaux[j]=NULL;
	}
	while(l!=NULL)
	{
		switch((l->valeur)[i])
		{
			case '0':sceaux[0]=ajoutQ(sceaux[0],l->valeur);
			break;
			case '1':sceaux[1]=ajoutQ(sceaux[1],l->valeur);
			break;
			case '2':sceaux[2]=ajoutQ(sceaux[2],l->valeur);
			break;
			case '3':sceaux[3]=ajoutQ(sceaux[3],l->valeur);
			break;
			case '4':sceaux[4]=ajoutQ(sceaux[4],l->valeur);
			break;
			case '5':sceaux[5]=ajoutQ(sceaux[5],l->valeur);
			break;
			case '6':sceaux[6]=ajoutQ(sceaux[6],l->valeur);
			break;
			case '7':sceaux[7]=ajoutQ(sceaux[7],l->valeur);
			break;
			case '8':sceaux[8]=ajoutQ(sceaux[8],l->valeur);
			break;
			case '9':sceaux[9]=ajoutQ(sceaux[9],l->valeur);
			break;
```

```
                        case 'A':sceaux[10]=ajoutQ(sceaux[10],l->valeur);
                        break;
                        case 'B':sceaux[11]=ajoutQ(sceaux[11],l->valeur);
                        break;
                        case 'C':sceaux[12]=ajoutQ(sceaux[12],l->valeur);
                        break;
                        case 'D':sceaux[13]=ajoutQ(sceaux[13],l->valeur);
                        break;
                        case 'E':sceaux[14]=ajoutQ(sceaux[14],l->valeur);
                        break;
                        case 'F':sceaux[15]=ajoutQ(sceaux[15],l->valeur);
                }
                temp=l;
                l=l->suivant;
                free(temp);
        }
        for(j=0;j<base;j=j+1)
        {
                 while(sceaux[j]!=NULL)
                {
                        l=ajoutQ(l,sceaux[j]->valeur);
                        temp=sceaux[j];
                        sceaux[j]=sceaux[j]->suivant;
                        free(temp);
                }
                free(sceaux[j]);
        }
        return l;
}

int main()
{
        int i,maxchiffres,base;
        liste l;
        char nb[30];
        l=(element*)malloc(sizeof(element));
        l=acquisition(&base, &maxchiffres);

        for(i=maxchiffres-1;0<=i;i=i-1)
        {
                l=tri(l,i,base);
        }
        affichage(l);

        getchar();
        return 0;
}
```