ÉCOLE CENTRALE DE LILLE
Algorithmique Avancée et Programmation

TEA AAP - TP2 2021

Types Abstraits de Données, Piles et Listes chaînées

Bruno SOARES ZIMMER

LILLE, FRANCE
2021

INTRODUCTION

Les vecteurs sont les formes les plus primordiales de création de liste, cette liste est appelée liste statique, puisque la taille est prédéfinie par le programmeur. Dans l'ordre, nous avons des vecteurs dynamiques, dans lesquels sa taille est définie dynamiquement, mais une fois qu'elle est allouée, il n'est plus possible de changer sa taille (seulement qu'elle vide tout de la mémoire et l'alloue à nouveau).

Enfin, nous avons le modèle le plus courant utilisé par les programmeurs du langage C, une liste créée de manière chaînée. Il a la particularité d'occuper la mémoire au fur et à mesure des besoins, tout comme il est possible de retirer des termes du milieu, et de le vider de sa mémoire (1).

Les objectifs de ces exercices sont multiples :

1) Nous devons appliquer le concept de pile utilisant la bibliothèque développée en classe.
2) Nous est demandé de créer 5 nouvelles fonctions à la manipulation de la liste, étant encore nécessaire de les développer sous forme récursive et itérative.
3) Il est demandé au programme de créer une image et cette image fournit tous les éléments de la liste liée.
4) L'exercice BONUS vise à créer une nouvelle gestion de liste, étant nécessaire de créer 4 nouvelles fonctions pour la gestion de celle-ci.

EXERCICE 1

Dans ce premier exercice, nous devons résoudre le problème de l'échange de bases sur une base décimale.
La solution est dans l'ordre

```
321  T_list changeBase(int value, int base, T_list s){
322      while(value>0){ // Ajouter chaque element de la base nouveau
323          push(value%base, &s);
324          value /= base; // value = value / base
325      }
326
327      return s;
328  }
```

Avec une fonction de teste dans l'ordre:

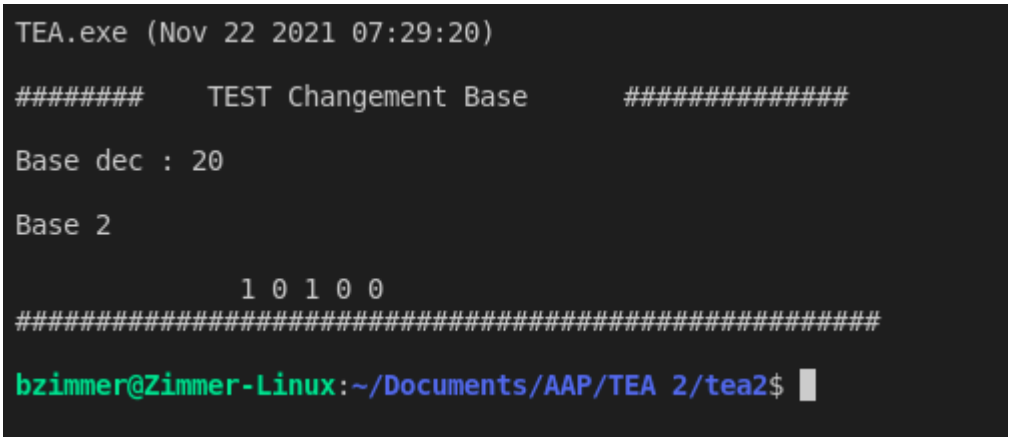
```
void testChangementBase(int value, int base, T_list s){
    printf("########     TEST Changement Base     ##############");NL();NL();
    printf("Base dec : %d\n", value );
    s = changeBase(value, base, s);NL();
    printf("Base base \n\n", base );
    printf("             ");
    showStack(&s);NL();
    printf("#####################################################");NL();NL();
}
```

```
13  int main(int argc, char ** argv) {
14
15      T_stack s = NULL;
16
17      CLRSCR();
18      WHOAMI();
19      if(argv[1] && argv[2])
20          testChangementBase(atoi(argv[1]), atoi(argv[2]), s);
21      else if(argv[1] && !argv[2])
22          testChangementBase(atoi(argv[1]), 2, s);
23      // testPNG();
24      // testGetSize();
25      //testTaillAdd();
26      //testSortAdd();
27      //testInList();
28      // testRemDup();
29
30      return 0;
31  }
32
```

Étaient la fonction "testChagementBase" une fonction pour tester la fonction "changeBase" avec nombres aléatoires.

La fonction changeBase a qu'une boucle que arrété quand le value atteint zéro, qui ça veut dire que il n'y a plus de division, suivant la formule a chaque division du valeur par la base on va enregistrer le reste de la division dans la pile.

Après nous montrerons la liste de la formule showStack, avec le resultát:



EXERCICE 2

2.1 Fonction “getSize”

La première fonction est appelée "getSize", en version récursive en utilisant le conditionnel ternaire:

```
160  unsigned int getSize(const T_list l)
161  { // Chercher dans la pile jusq'au fin et après par rec. compter
162  //Renvoyer la taille d'une liste
163      /*
164      if(l != NULL){
165          return getSize(l->pNext) +1
166      }
167      if l == NULL{
168          return 0
169      }
170      */
171      return l ? getSize(l->pNext) + 1 : 0;
172  }
```

Avec une fonction de teste dans l’ordre:

```
35  void testGetSize(void){
36      T_stack s = NULL;
37      int i;
38      printf("########     TEST Get Size     ##############");NL();NL();
39      for(i=10; i>0;i--){
40          push(i,&s);
41      }
42      showStack(&s);NL();
43      printf("%u \n", getSize(s));
44
45      printf("######################################################");NL();NL();
46  }
```

La fonction essentiellement fait la récursivité jusqu’au dernier élément et après retourne tout en augmentant un a chaque interaction de la fonction.

Le résultat:

```
TEA.exe (Nov 22 2021 17:39:59)

########     TEST Get Size      ##############

1 2 3 4 5 6 7 8 9 10
10
############################################################

bzimmer@Zimmer-Linux:~/Documents/AAP/TEA 2/tea2$
```

2.2 Fonction “tailAddNode”

Le code pour ces fonctions est dans l'ordre:

```
174  T_list tailAddNode(T_elt e, T_list l){//insérer un élément en queue d'une liste chaînée
175  /*  if(l != NULL )
176      |   if(l->pNext != NULL) return  tailAddNode(e, l->pNext);
177      |   else return l->pNext = newNode(e)
178      else
179      |   return newNode(e);*/
180      if(l)
181      |   return l->pNext ? tailAddNode(e, l->pNext) : (l->pNext = newNode(e));
182      else
183      |   return newNode(e);
184  }
```

Le principe de la première fonction est assez simple, faites défiler toute la liste chaînée jusqu'au dernier terme, dès que vous le trouvez, un nouvel élément est ajouté à la liste. Pour vérifier le fonctionnement il y a la fonction de test est dans l’ordre:

```
48  void testTaillAdd(void){
49      T_stack s = NULL;
50      int i;
51      printf("########    TEST Add Tail      ##############");NL();NL();
52
53      for(i=10; i>0;i--){ // Crée une liste ordinaire
54          push(i,&s);
55      }
56      showStack(&s);NL();NL();
57      printf("Maintenant ajouter '5'");NL();NL();
58      tailAddNode(5, s); // utiliser la fonction
59      showStack(&s);NL();NL();
60
61
62      printf("##################################################");NL();NL();
63  }
```

Le résultat:

```
TEA.exe (Nov 22 2021 17:39:59)

########    TEST Get Size      ##############

1 2 3 4 5 6 7 8 9 10
10
############################################################

bzimmer@Zimmer-Linux:~/Documents/AAP/TEA 2/tea2$
```

2.3 Fonction “sortAddNode”

Suivre, le code de cette fonction:

```
193  T_list sortAddNode(T_elt e, T_list l){
194      //Insérer un élément dans une liste chaînée triée par ordre croissant à la bonne place
195      //Que faut-il ajouter à notre module de gestion de T_elt pour permettre de comparer deux
196      //T_elt ?  Réaliser les modifications nécessaires
197      T_node *newN;
198      if (compElt(e, l->data)>=0 && l->pNext)
199      {
200          return sortAddNode(e, l->pNext);
201      }else if(compElt(e, l->data)>=0 && !l->pNext){
202          newN = newNode(e);
203          newN->pNext = l->pNext;
204          l->pNext = newN;
205          return l;
206      }
207      else
208      {
209          newN = newNode(l->data);
210          newN->pNext = l->pNext;
211          l->pNext = newN;
212          l->data = e;
213          return l;
214      }
215  }
```

Le principe est essentiellement de passer par chaque vecteur, et à chaque avance, de faire une comparaison entre cet élément de la liste, et l'élément à placer. La fonction compElt est utilisée pour comparer. Le code de cette fonction est en séquence. La fonction comElt a trois formes définies par le variables: ELT_INT, ELT_CHAR, ELT_STRING, de la forme:

```
74   #ifdef ELT_STRING
75   int compElt(T_elt e1, T_elt e2) {
76       return (strcmp(e1, e2)<0) ? 0 : 1
77   }
```

```
45   #ifdef ELT_INT
46   int compElt(T_elt e1, T_elt e2) {
47       /*
48       if e1<e2
49           -1
50       if e1>e2
51           1
52       e1 ==e2
53       0
54       */
55       return (e1<e2)?-1:(e1>e2?1:0);
56   }
```

```
19   #ifdef ELT_CHAR
20   int compElt(T_elt e1, T_elt e2) {
21       return (e1<e2)?0:1;
22   }
```

Le principe est le même que la fonction "sortAddNode", cependant, au lieu

d'utiliser la boucle, nous utilisons la récursion pour faire défiler toute la liste.
Les résultats de ces deux demandes sont présentés dans l'ordre suivant.

Le dossier de test est ici:

```
65  void testSortAdd(void){
66      T_stack s = NULL;
67      int i;
68      printf("########    TEST Sort Add      ##############");NL();NL();
69      
70      for(i=10; i>0;i--){ // Crée une liste ordinaire
71          push(i,&s);
72      }
73      showStack(&s);NL();NL();
74      printf("Maintenant ajouter '5'");NL();NL();
75      sortAddNode(5, s);// utiliser la fonction avec differents values
76      showStack(&s);NL();NL(); // afficher le resultat
77      printf("Maintenant ajouter '13'");NL();NL();
78      sortAddNode(13, s);
79      showStack(&s);NL();NL();
80      printf("Maintenant ajouter '0'");NL();NL();
81      sortAddNode(0, s);
82      showStack(&s);NL();NL();
83
84      
85      printf("#####################################################");NL();NL();
86  }
```

Le résultat:

```
TEA.exe (Nov 22 2021 17:41:32)

########    TEST Sort Add      ##############

1 2 3 4 5 6 7 8 9 10

Maintenant ajouter '5'

1 2 3 4 5 5 6 7 8 9 10

Maintenant ajouter '13'

1 2 3 4 5 5 6 7 8 9 10 13

Maintenant ajouter '0'

0 1 2 3 4 5 5 6 7 8 9 10 13

#####################################################

bzimmer@Zimmer-Linux:~/Documents/AAP/TEA 2/tea2$
```

2.4 Fonction “inList”

Le code pour cette fonction est dans l'ordre.

```
217  int inList(T_elt e, const T_list l)
218  {
219      //Vérifier si un élément appartient à une liste chaînée
220      return (l && (e == l->data)) ? 1 : (l->pNext &&(e != l->data)) ? inList(e, l->pNext) : 0;
221      // if (l && (e == l->data)){
222      //     return 1;
223      // }else if (l->pNext && (e != l->data)){
224      //     return inList(e, l->pNext);
225      // }else{
226      //     return 0;
227      // }
228  }
```

L'idée de base des deux fonctions est de vérifier toute la liste, jusqu'à ce que vous trouviez un élément similaire au paramètre précédent. Dans le premier cas, la liste est recherchée de manière itérative, et dans le second de manière récursive.

Le dossier de test est ici:

```
87   void testInList(void){
88       T_stack s = NULL;
89       int i;
90       printf("########     TEST In List      ##############");NL();NL();
91       for(i=10; i>0;i--){ // Crée une liste ordinaire
92           push(i,&s);
93       }
94       showStack(&s);NL();NL();
95       printf("Maintenant ajouter '5'");NL();
96       if(inList(5, s))// utiliser la fonction avec differents values
97           printf("5 esta na lista \n\n\n"); // afficher le resultat
98       else
99           printf("5 não esta na lista \n\n\n");
100      printf("Maintenant ajouter '13'");NL();
101
102      if(inList(13, s))
103          printf("13 esta na lista \n\n\n");
104      else
105          printf("13 não esta na lista \n\n\n");
106
107      printf("##################################################");NL();NL();
108  }
```

Le résultat:

```
TEA.exe (Nov 22 2021 17:42:00)

########     TEST In List      ##############

1 2 3 4 5 6 7 8 9 10

Maintenant ajouter '5'
5 esta na lista


Maintenant ajouter '13'
13 não esta na lista


##################################################

bzimmer@Zimmer-Linux:~/Documents/AAP/TEA 2/tea2$
```

2.5 Fonction “removeDuplicates”

Avant d'afficher le code de la fonction, l'extrait de code suivant montre le code d'une fonction auxiliaire ‘showDuplicates“, qui a la fonctionnalité d'afficher un texte sur le terminal grâce à la création d'une constante.

```
#ifdef DEBUG
    void showDuplicates(T_elt e){
        printf("Duplicate: %s\n", toString(e));
    }
#else
    void showDuplicates(T_elt e){}
#endif
```

Ainsi, si l'indicateur a défini la constante "DEBUG", la fonction "showDuplicates" affichera un texte, et sinon, rien ne sera affiché.

La fonction "removeDuplicates" se déroule dans l'ordre.

```
T_list removeDuplicates(T_list l){
    //Éliminer les doublons d'une liste chaînée tout en libérant la mémoire devenue inutile.
    //Cette fonction doit afficher quels éléments sont supprimés si la constante symbolique
    //DEBUG est déclarée dans le fichier list.h.
    T_list lAux = NULL, lAuxToDel = NULL;
    if (l){
        lAux = l;
        while (lAux->pNext){//while pour rechercher le comparison
            if (compElt(l->data, (lAux->pNext)->data) == 0){
                showDuplicates((lAux->pNext)->data);
                if ((lAux->pNext)->pNext){//moyen de la liste avec duplicate
                    lAuxToDel = lAux->pNext;
                    lAux->pNext = lAuxToDel->pNext;

                    free(lAuxToDel);
                }
                else{
                    free(lAux->pNext);//fin de la liste
                    lAux->pNext = NULL;
                    break;
                }
            }else{
                lAux = lAux->pNext;
            }
        }
        removeDuplicates(l->pNext); //le deuxième boucle pour rechercher nombre
    }
    return l;
}
```

Le principe de cette fonction est de parcourir l'ensemble de la liste, et dans chaque élément, de faire une vérification avec le reste de la liste, afin de connaître l'existence d'éléments égaux. Si c'est le cas, la fonction désactive de mémoire l'élément en double et réactive les liens avec la liste.

La fonction de test:

```
108  void testRemDup(void){
109      T_stack s = NULL;
110      int i;
111      printf("########     TEST Remove Duplicate      ##############");NL();NL();
112      for(i=5; i>0;i--){ // Crée une liste ordinaire
113          push(i,&s);
114          push(5,&s);
115      }
116      printf("Liste");
117      showStack(&s);NL();NL();
118      printf("Maintenant retirer '5'");NL();NL();
119      s = removeDuplicates (s);// utiliser la fonction
120      showStack(&s);NL();NL(); // afficher le resultat
121      printf("#################################################");NL();NL();
122  }
```

Et le résultat:

```
TEA.exe (Nov 22 2021 17:45:55)

########     TEST Remove Duplicate      ##############

Liste5 1 5 2 5 3 5 4 5 5

Maintenant retirer '5'

5 1 2 3 4

#################################################

bzimmer@Zimmer-Linux:~/Documents/AAP/TEA 2/tea2$
```

EXERCICE 3

L'objectif de cet exercice est de créer un fichier .dot contenant les informations d'une liste chaînée, et de convertir ensuite ce fichier en fichier .png. En principe, de la même manière que vous interagissez avec des fichiers .txt en C, vous manipulez des fichiers .dot. En utilisant des fonctions telles que "fopen", "fscanf", "fprintf".

La première fonction développée a été "generateHeaderPNG", qui vise à créer une le header du document .dot. Son code est séquentiel.

```
290  void generateHeaderPNG(FILE * filePNG)
291  {
292      char ch, sourceFile[30];
293      FILE *source;
294
295      strcpy(sourceFile, "./Headers/header.dot");
296
297      source = fopen(sourceFile, "r");
298      while( ( ch = fgetc(source) ) != EOF )
299          fputc(ch, filePNG);
300
301      //printf("Header copied successfully.\n");
302
303      fclose(source);
304  }
```

Dans la fonction principale, un texte est affiché dans le terminal, annonçant la fin de la création du fichier. Dans l'un des processus, la fonction "generatePNG" est appelée, qui est présentée en séquence.

```
267  void generatePNG(const T_list l, const char * filename){
268      FILE *file;
269
270      T_list aux = l;
271      int i;
272
273      file = fopen(filename, "w");
274      generateHeaderPNG(file);
275
276      for(i=1;aux;i++, aux = aux->pNext)
277      {
278          if(aux->pNext){
279              fprintf(file, "\"ID_%04d\" [label = \"{<elt> %s | <next> }\"];\n", i,toString(aux->data));
280              fprintf(file, "\"ID_%04d\" : next -> \"ID_%04d\";\n", i,i+1);
281          }else{
282              fprintf(file, "\"ID_%04d\" [label = \"{<elt> %s | <next> NULL }\"];\n", i,toString(aux->data));
283          }
284      }
285
286      fputs("}", file);
287  }
```

Cette fonction a les principes suivants : créer le fichier, appeler la fonction "generateHeaderPNG" (une fonction qui crée l'en-tête par défaut du fichier .dot), recevoir la liste, et faire défiler la liste entière. Pendant que la fonction fait défiler la liste, à chaque processus, elle crée une ou deux nouvelles lignes dans le code, selon qu'il s'agit d'un élément du milieu ou de la fin de la liste. Chaque ligne qui est placée dans le fichier final est composée de parties très similaires.

Le dossier de test est:

```
9   void testPNG(void){
10      printf("########    TEST PNG     ##############");NL();NL();
11      T_stack s = NULL;
12      int i;
13
14      for(i=10; i>0;i--){// Crée une liste ordinaire
15          push(i,&s);
16      }
17      showStack(&s);NL();
18      generatePNG(s, "./Output/listPNG.dot");// utiliser la fonction
19
20      printf("#####################################################");NL();NL();
21
22  }
```

Et le résultat:

CONCLUSION

Après tous ces développements, il est possible de remarquer la grande évolution que nous avons eue par rapport à la gestion des listes chaînées. Il existe maintenant une variété de fonctions déjà définies et extrêmement utiles dans l'utilisation des listes.