Travaux Dirigés de langage C - DUT 1^{ere} année

Chapitre 1

Syntaxe de base

1.1 Test d'un générateur de nombre aléatoire

La bibliothèque standard du langage C offre au programmeur plusieurs fonctions pour générer des nombres aléatoires. La plus connue est `rand()`. Vous allez écrire un programme `verifrand.c` qui estime la répartition des nombres aléatoires générés : moyenne et variance.

Générez 1 million (`NBTIRAGE`) notes (x_i) entre 0 et 20 (`N`) comprises (`NB_NOTE` = 21) à l'aide de la fonction `rand()`. Répartissez-les dans le tableau effectif (`int n[NB_NOTE]`) où $n[x_i]$ représente l'effectif (nombre d'occurrences cumulées) de la note x_i.

- calculez et affichez la moyenne arithmétique : $moy = \frac{1}{NBTIRAGE} \sum_0^N i * n[i]$;
- calculez et affichez la variance de la série : $var = \frac{1}{NBTIRAGE} \sum_{k=0}^N k^2 * n[k] - moy^2$

1.2 Conjugaison d'un verbe régulier du premier groupe

Ecrire un programme C qui lit au clavier un verbe régulier du premier groupe, le vérifie (terminaison en er) et le conjugue au présent de l'indicatif sans utiliser de boucle puis en utilisant une boucle et les tableaux de chaînes à deux dimensions.

1.3 Participe présent des verbes du deuxième groupe

Les verbes du deuxième groupe sont les verbes en *-ir* dont le participe présent est en *-issant*. ex : bondir > bondissant.

Ecrire en `C` la fonction `void partPres(char verbe2[], char pp[])` qui prend dans `verbe2` un verbe du deuxième groupe et lui calcule son participe présent dans `pp`.

1.4 Evaluer les expressions suivantes

En supposant

```
A=20   B=5   C=-10   D=2   X=12   Y=15
```

Calculez la valeur des expressions suivantes et les nouvelles valeurs des variables dont le contenu a changé. Faites le calcul sans utiliser l'ordinateur puis véfifier le sur votre machine.

```
(1)   (5*X)+2*((3*B)+4)
(2)   (5*(X+2)*3)*(B+4)
(3)   A == (B=5)
(4)   A += (X+5)
(5)   A != (C *= (-D))
(6)   A *= C+(X-D)
(7)   A %= D++
(8)   A %= ++D
(9)   (X++)*(A+C)
(10)   A = X*(B<C)+Y*!(B<C)
(11)   !(X-D+C)||D
(12)   A&&B||!0&&C&&!D
(13)   ((A&&B)||(!0&&C))&&!D
(14)   ((A&&B)||!0)&&(C&&(!D))
```

solutions sur http ://www.ltam.lu/cours-c/homesol.htm ou sur votre machine.

1.5 Utilisation de l'instruction switch

Ecrire un programme C qui lit un nombre hexadécimal sur 2 chiffres et écrit sa valeur à l'écran. Utiliser l'instruction `scanf("%c%c", &c1, &c2);` et les déclarations `char c1,c2;`.

1.6 Conversions de chaînes en nombres

1.6.1 Exercice

Ecrire un programme `main()` qui lit une chaîne de caractères pris entre '0' et '9' et imprime leur somme.

```
./ex
89
17
./ex
32
5
```

1.6.2 Exercice

Ecrire une procédure qui lit une chaîne de caractères pris entre '0' et '9' et l'interprète comme un entier positif dans la base décimale. Pour la conversion, utiliser les fonctions de `<ctype>` [1] et la relation d'ordre alphabétique des caractères '0' à '9'. Mémoriser le résultat dans une variable du type `long`. La conversion s'arrête à la rencontre du premier caractère qui ne représente pas de chiffre décimal. Utiliser un indicateur logique `OK` qui précise si la chaîne représente correctement une valeur entière et positive.

1.6.3 Exercice

Ecrire une procédure qui lit une chaîne de caractères et l'interprète comme un entier positif dans la base hexadécimale. Pour la conversion, utiliser la fonction `isdigit()` de `<ctype>` et la précédence alphabétique des caractères. La conversion ignore les caractères qui ne représentent pas de chiffre hexadécimal et s'arrête à la fin de la chaîne de caractères. Le résultat sera mémorisé dans une variable du type `long` et affiché dans les bases hexadécimale et décimale.

1.6.4 Le cas de nombres réels

On considère la fonction suivante qui convertit une chaîne `C` de caractères (par exemple `char t[] = ‘‘123’’ ;`) dans une variable numérique (par exemple `int x ;`).

```
#include <stdio.h>
#include <stdlib.h>
int ascInt(char t[])
/* conversion chaine -> valeur */
{
  char cx ;
  int i = 0 ;
```

1. taper “ctype” sur un moteur de recherche

```
  int x = 0 ;
  cx = t[0] ;
  while (cx != '\0') {
    if (cx <= '9' && cx >= '0') x = x * 10 + (cx - '0') ;
    i++ ;
    cx = t[i] ;
  }
  return x ;
}
```

Sur la base de cette fonction, écrire en `C` la fonction `float ascFloat(char t[])` qui convertit une chaîne `C` (par exemple `char t[] = ``123.45'' ;`) dans une variable numérique (par exemple `float x ;`). Les chaînes peuvent (ou non) avoir des parties décimales et entières, par exemple : ``123.45'', ``123'', ``123.'', ``.45''.

1.6.5 Conversion d'un nombre en chaîne

Sans utiliser de fonctions de la bibliothèque `C`, écrire la fonction

```
int itoa(int val, char* s, int maxLen)
```

qui convertit la valeur `val` en une chaîne de caractères `s` en notation hexadécimale. Par exemple la valeur 123, dans val donne la chaîne "7B" dans s. Le paramètre `maxLen` donne la taille disponible dans `s` (cf. Tab. 1.1). Si la conversion est possible, la fonction retourne 0 sinon 1. Effectuez des divisions successives par 16 et transformez les restes – entre 0 et 15 – en un caractère correspondant grâce au tableau `char n2c[] = "0123456789ABCDEF" ;`.

1.7 Tableaux à plusieurs dimensions

Attention aux déclarations ! le langage C manipule très mal les tableaux à plusieurs dimensions car celles-ci ne font pas partie de la représentation du tableau. En particulier, il est très lourd de passer une matrice en paramètre d'une fonction.

1.7.1 Multiplication de matrices

En multipliant une matrice A de dimensions N et M avec une matrice B de dimensions M et P on obtient une matrice C de dimensions N et P : A(N,M) * B(M,P) = C(N,P)

```
int main()
{
  char buf1[80], buf2[2] ;
  int n = 1023 ;

  itoa(n, buf1, 80) ;
  printf("%s\n", buf1) ; /* imprime 3FF sur une ligne */

  if (itoa(n, buf2, 2) == 0)
    printf("%s\n", buf2) ; /* never get there */
  else
    printf("conversion impossible\n") ; /* imprime conversion impossible */
  return 0 ;
}
```

TABLE 1.1 – Deux appels de itoa

La multiplication de deux matrices se fait en multipliant les composantes des deux matrices lignes par colonnes :

$$c_{ij} = \sum_{k=1}^{M} a_{ik} * b_{kj}$$

Rappel :

$$\begin{pmatrix} a & b & c \\ e & f & g \\ h & i & j \\ k & l & m \end{pmatrix} * \begin{pmatrix} p & q \\ r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a*p+b*r+c*t & a*q+b*s+c*u \\ e*p+f*r+g*t & e*q+f*s+g*u \\ h*p+i*r+j*t & h*q+i*s+j*u \\ k*p+l*r+m*t & k*q+l*s+m*u \end{pmatrix}$$

Ecrire un programme qui effectue la multiplication de deux matrices A et B. Le résultat de la multiplication sera mémorisé dans une troisième matrice C qui sera ensuite affichée.

Refaire ce programme sous forme d'une fonction qui admet les matrices et leurs dimensions en paramètres. Les matrices sont de tailles quelconques mais compatibles.

Application : prendre

$$A = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}, \quad B = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

On doit trouver, pour n'importe quelle valeur de θ, $C = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

1.7.2 Révisions

Ecrire un programme qui vérifie la validité d'un numéro de CCP entré au clavier. Un numéro de CCP est composé de trois parties : un numéro de compte, un séparateur '-' et un numéro de contrôle. Un numéro de CCP est correct :

- si le reste de la division entière de la valeur devant le séparateur '-' par 97 est différent de zéro et égal à la valeur de contrôle ;
- si le reste de la division par 97 est zéro et la valeur de contrôle est 97.

Exemples : On entre au clavier les chaînes suivantes

- CCP 15742-28 : 15742 modulo 97 = 28 correct
- CCP 72270-5 : 72270 modulo 97 = 5 correct
- CCP 22610-10 : 22610 modulo 97 = 9 incorrect
- CCP 50537-0 : 50537 modulo 97 = 0 nombre incorrect, car la valeur de contrôle devrait être 97.
- CCP 50537-97 : 50537 modulo 97 = 0 correct

Ecrivez et utiliser une fonction `ccptest()` qui obtient comme paramètres les deux parties numériques d'un nombre de CCP et qui affiche alors un message indiquant si le numéro de CCP est valide ou non.

Chapitre 2

Les Pointeurs

Un pointeur est une variable qui contient une référence (sous forme d'une adresse en mémoire) à une autre variable.

2.1 Exercice

Soit P un pointeur qui 'pointe' sur un tableau A :

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses fournissent ces expressions ? Faites l'exercice dans un premier temps sans machine puis vérifiez vos résultats grâce à un programme `C` qui imprime les valeurs des expressions. Quand l'expression est une adresse (par ex. `&i`) , imprimez la valeur de la variable référencée (par ex. `*(&i)`) et comparez (par ex. à `i`).

1. *P+2
2. *(P+2)
3. &P+1
4. &A[4]-3
5. A+3
6. &A[7]-P
7. P+(*P-10)
8. *(P+*(P+8)-A[7])

2.2 Exercice

– Quelles sont les valeurs des variables `a` et `t` après l'exécution du programme ci-dessous (Tab. 2.1) ?

```
1  int main()
2  {
3    int t[2] ;
4    int a, *b, *c ;
5    t[0] = 3 ;
6    t[1] = 5 ;
7    a = 10 ;
8    b = t ;
9    c = &a ;
10   *c = *b ;
11   a++ ;
12   b++ ;
13   *b = 2 * a ;
14 }
```

TABLE 2.1 – Quelles sont les valeurs de a et t à la fin de l'exécution de ce programme ?

– Simuler l'exécution de ce programme (cf. Tab. 2.2) :

```
1  #include <stdio.h>
2  int main()
3  {
4      int a;
5      int *x,*y;
6
7      a = 90;
8      x = &a;
9      printf("*x = %d\n",  *x) ;
10
11     *x = 100;
12     printf("a vaut : %d\n",  a) ;
13
14     y = x;
15     *y = 80;
16     printf("a vaut : %d\n",  a) ;
17
18     return 0;
19 }
```

TABLE 2.2 – Remarquez comment `a` est modifiée par le biais des pointeurs `x` et `y`

– Qu'imprime le programme suivant (cf. Tab. 2.3) ?

2.3 Fonction qui calcule plusieurs résultats

Ecrire une fonction `void minmax (int i, int j, ...)` qui calcule le minimum et le max de i et j. Utilisez un passage de paramètres par adresse. Testez `minmax()` dans un programme principal.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
  char *message ;
  char *dm[3] ;
  message = (char *) malloc(35) ;
  strcpy(message, "Belle Marquise, vos beaux yeux ...") ;
  message[19] = '\0' ;
  message[14] = '\0' ;
  dm[0] = message ;
  dm[2] = message + 20 ;
  dm[1] = message + 15 ;
  printf("%s %s %s\n", dm[1], dm[0], dm[2]) ;
  printf("%s %s %s\n", dm[0], dm[1], dm[2]) ;
  free(message) ;
  return 0 ;
}
```

TABLE 2.3 – Que dit monsieur Jourdain ?

2.4 Tableau de compteurs

Ecrire un programme qui lit une chaîne de caractères CH au clavier et qui compte les occurrences des lettres de l'alphabet en ne distinguant pas les majuscules et les minuscules. Utiliser un tableau ABC de dimension 26 pour mémoriser le résultat et un pointeur PCH pour parcourir la chaîne CH et un pointeur PABC pour parcourir ABC. Afficher seulement le nombre des lettres qui apparaissent au mois une fois dans le texte.

Exemple :

```
Entrez un ligne de texte (max. 100 caractères) :
Jeanne
La chaîne "Jeanne" contient :
1   fois la lettre 'A'
2   fois la lettre 'E'
1   fois la lettre 'J'
2   fois la lettre 'N'
```

2.5 Tableaux de pointeurs

Exercice

Refaire l'exercice de la conjugaison d'un verbe du premier groupe avec deux tableaux de pointeurs : `pronoms[]` et `terminaisons[]` que vous initialiserez dans la déclaration.

```
char *pronoms[] = { ... } ;
```

```
char *terminaisons[] = { ... } ;
```

2.5.1 Arguments de la fonction main()

Ecrire la commande `echo` du système Unix. Ce programme ne gérera que l'option `-n` qui supprime le passage à ligne.[1] Testez le avec la commande : `monEcho *`. Que déduisez vous ?

2.6 Ajouter / supprimer / modifier des composantes d'un nom de fichier

Réaliser sous forme de procédures C des fonctions utilitaires pour ajouter, supprimer, modifier des extensions ou des répertoires de noms de fichier.

```
char *chgext(char *name, const char *ext)
/* change the extension of NAME with EXT
   UNIX Version.  return modified name */

char *chgdir(char *name, const char *dir)
/* change the directory of NAME with DIR
   Version UNIX. return modified name */
```

Mettez en œuvre (c.-a.-d. testez) ces procédures en utilisant un programme main() simple (cf. `test-ext` et `test-dir` dans l'exemple ci-dessous).

```
Src$ ./test-ext /Users/jfmari/Src/a.out .exe
/Users/jfmari/Src/a.exe
Src$ ./test-dir /Users/jfmari/Src/a.out /tmp/
/tmp/a.out
Src$
```

2.7 Affectation de chaînes en Java

Ecrire en `C` sans utiliser les fonctions de `string.h`, deux versions de la fonction `strcpy(char s1[], char s2[])`. Cette fonction copie s2 dans s1, elle se traduit par `s1 = s2` en Java.

1. Appelez le monEcho afin qu'il ne soit pas confondu avec la fonction Unix du même nom

version 1 : `void strcpy(char s1[], char s2[]).` On suppose que le tableau s1 est assez grand pour recevoir s2 ;

version 2 : `char* strcpy(char* s1, char* s2).` On suppose que s1 et s2 sont deux pointeurs vers des chaînes de caractères allouées grâce à la fonction `malloc`. Ecrire une version qui copie s2 dans s1 et retourne l'adresse de s1 (qui peut être nouvelle dans le cas où un nouvel emplacement a été alloué parce que la taille de s2 était supérieure à la taille initialement allouée pour s1).

2.8 Pointeurs et chaînes de caractères

1. Écrire une fonction qui admet en paramètre une chaîne contenant 5 mots, séparés par des espaces et qui les affiche ensuite dans une ligne, mais dans l'ordre inverse. Les mots sont mémorisés dans 5 variables `m1`, ... , `m5`.

   Utiliser cette fonction pour traiter un fichier texte. Pour lire une ligne contenant des espaces, utilisez la fonction `char* fgets(char* s, int n, FILE* stream)` en donnant la valeur `stdin` à la variable `stream`.

   ```
   char buffer[80] ;
   ...
   fgets(buffer, 80, stdin) ; /* lecture au clavier dans buffer */
   ```

   Donnez deux versions : une qui utilise la fonction C `strtok()` définie dans `string.h` et une autre sans utiliser `strtok()`.

2. A partir d'un dictionnaire (fichier contenant les couples (mot, prononciation) avec une prononciation par mot et par ligne, ce fichier est trié selon l'ordre alphabétique des mots) et d'un texte quelconque représenté dans un fichier texte, phonétiser le texte, c'est à dire, créer un nouveau fichier contenant le texte où les mots du dictionnaire sont remplacés par leur prononciation. Si un mot n'est pas dans le dictionnaire le laisser tel quel et faire un message d'erreur à l'écran. Comme dans l'exercice précédent, utiliser `fgets(buffer, ..)` pour lire une ligne du fichier dans laquelle apparaissent des espaces.

   Exemple de dictionnaire :

   ```
   cent     s an
   comme    k o m
   france   f r an s
   vingt    v in
   ```

   Exemple de texte source : `Il y a cent vingt ans en République de France`

   Exemple de résultat : `Il y a s an v in ans en République de f r an s`

2.9 Debugger

Dans le programme wordcnt.c[2], des bugs ont été introduits afin que vous appreniez à utiliser le "debugger" gdb. Commencez par enlever tous les avertissements (*warning*) puis corrigez les 2 bugs.

```
1  /*****
2  *  Lit un fichier texte ; compte le nombre de mots de longueur
3  *  1, 2, 3, etc.
4  *  REMARQUE : ce programme est à utiliser pour l'apprentissage du
5  *             débogueur.
6  *             Des bogues ont été introduites intentionnellement.
7  ******/
8  #include <stdio.h>
9  #include <ctype.h>
10 #include <string.h>
11 #include <stdlib.h>
12
13 #define MAXWORDLEN      16
14 #define NUL             ((char)0)
15 #define SPACE           ((char)0x20)
16 /*****
17 *  Trouve le mot suivant dans le tampon de ligne.
18 *  EN ENTREE :        wordptr pointe sur le premier caractère
19 *                     d'un mot ou l'espace précédent.
20 *  VALEUR RETOURNEE : un pointeur sur le premier caractère du mot.
21 *                     S'il n'y a pas d'autres mots, un pointeur sur
22 *                     le caractère nul d'arret.
23 *****/
24 char *nextword(char *wordptr)
25 {
26 /* Avance jusqu'au prochain caractère qui n'est pas un espace. */
27   while ( *wordptr == SPACE )
28     wordptr++;
29   return wordptr ;
30 }
31 /*****
32 *  Trouve la longueur d'un mot. Un mot est une suite de caractères
33 *  terminée par un espace ou un caractère nul.
34 *  EN ENTREE :         wordptr pointe sur un mot.
35 *  VALEUR RETOURNEE : la longueur du mot.
36 *****/
37 int wordlen(char *wordptr)
38 {
39   char *wordlimit;
40
41   wordlimit = wordptr;
42   while ( *wordlimit  && *wordlimit != SPACE )
43     wordlimit++;
44   return wordlimit - wordptr ;
45 }
46 /*****
47 * La fonction principale.
48 *****/
49 int main(int argc, char* argv[])
50 {
51   FILE  *infile;         /* Fichier en entrée de nom dans argv[] */
52   char  linebfr[1024],   /* Tampon de ligne en entrée, */
53                          /* très long par sécurité. */
54         *wordptr;        /* Pointeur sur le mot suivant de linebfr. */
55   int     i;             /* Variable utilitaire. */
56   int   wordlencnt[MAXWORDLEN],
57                          /* Les longueurs de mot sont comptées dans
58                             les éléments de 1 à MAXWORDLEN. L'élément
59                             0 n'est pas utilisé. Le tableau est
60                             statique de sorte que les éléments n'ont
61                             pas à etre mis à zéro au moment de
62                             l'exécution. */
63               overlencnt;
```

2. http ://www.loria.fr/~jfmari/Cours/wordcnt.c

```
64                          /* Les mots trop longs sont comptés ici. */
65    if (argc < 2) {
66      printf("Il faut spécifier un fichier d'entrée !\n");
67      exit(1);
68    }
69    strcpy(linebfr, argv[1]) ;
70
71    for (i = 0 ; i < MAXWORDLEN ; i++) wordlencnt[i] = 0 ;
72    infile = fopen( linebfr, "r" );
73    if ( !infile ) {
74      perror(argv[1] );
75      exit(1);
76    }
77  /* Chaque boucle traite une ligne. REMARQUE : Lorsque la longueur
78     d'une ligne dépasse la capacité du tampon d'entrée, le programme
79     peut produire des résultats incorrects. Ceci devient improbable
80     avec un très grand tampon. */
81
82    while (   fgets( linebfr, sizeof(linebfr), infile )   )   {
83      /* printf("%s\n",linebfr); */
84      /* Vérifie qu'il n'y a pas débordement de capacité du tampon
85         et supprime le caractère d'interligne final. */
86      i = strlen(linebfr);
87      if ( linebfr[i-1] != '\n' )
88        printf("Ligne trop longue commençant par :\n\t%70s\n", linebfr);
89      else
90        linebfr[i-1] = NUL;
91
92      /* wordptr pointe sur le premier mot de linebfr (après les espaces
93         du début). */
94      wordptr = nextword(linebfr);
95
96      /* Chaque boucle traite un mot. La boucle se termine lorsque [nextword]
97         retourne un NULL, indiquant qu'il ne reste plus de mots. */
98      while (*wordptr) {
99      /* Trouve la longueur de ce mot, incrémente l'élément approprié
100        du tableau des compteurs de longueur et pointe sur l'espace
101        suivant le mot. */
102       i = wordlen(wordptr);
103       if ( i > MAXWORDLEN )
104         overlencnt++;
105       else
106                                          ;
107       wordlencnt[i]++;
108     wordptr += i;
109
110       /* Trouve le mot suivant (s'il existe). */
111       wordptr = nextword(wordptr);
112     }
113   }
114 /* Affiche les compteurs de longueurs des mots. Chaque boucle
115    en imprime un. */
116   printf("%s\n", " Comptage des longueurs");
117   for ( i=1; i < MAXWORDLEN; i++ )
118     printf( "  %5d %5d\n", i, wordlencnt[i] );
119   printf( "  Mots plus longs %5d\n", overlencnt );
120
121 /* Ferme le fichier et quitte. */
122   fclose(infile);
123   return 0 ;
124 }
```

2.10 Traduction de Java vers C

2.10.1 PileString

On souhaite traduire en `C` le programme Java donné table 2.4. Ce programme manipule 2 piles de chaînes de caractères. Pour traduire en `C` les classes Java `String` et `PileString`, vous utiliserez les types de données `MString` et `PileString` définis ci-dessous :

```
typedef struct pile_string {
  MString *tab ; // tab[0], tab[1], ... sont les chaines de la pile
  int sommet ; //  de la pile
  int sommetMax ; // de la pile
} PileString ;
```

et

```
typedef char* MString ; // le type chaine de caracteres
```

2.11 Fichiers et pointeurs

Nous disposons d'un fichier texte qui indique le nom, le prénom et le groupe de chaque étudiant (cf. Tab 2.5).

A partir de la méthode de tri par permutation (appelée aussi tri à bulles) donnée page 37 du poly, on demande de créer un fichier trié à partir de `etudiant.txt` de plusieurs façons différentes.

2.11.1 Allocation statique

Les étudiants sont rangés dans un tableau de structures où chaque chaîne de caractères fait 80 caractères de long.

2.11.2 Allocation dans des tableaux dynamiques

les étudiants sont rangés dans un tableau dont on calcule la taille après avoir compté le nombres d'étudiants dans le fichier. Chaque étudiant est représenté par le type `Etudiant1` donné Tab. 2.6.

```java
import java.util.*;

public class PileString // une pile de chaines de caracte'res
{
    private String tab[] = null ;
    private int sommet = 0 ;
    private int sommetMax ;
    public PileString(int tailleMax)
    {
        tab = new String[tailleMax] ;
        sommet = 0 ; // premiere case libre au dessus de la pile
        sommetMax = tailleMax ;
    }
    public void empiler(String v)
    {
        if (sommet < sommetMax) {
            tab[sommet] = v ;
            sommet++ ;
        }
    }
    public String depiler()
    {
        String v = "" ;
        if (sommet > 0) {
            sommet-- ;
            v = tab[sommet] ;
        }
        return v ;
    }
    public void afficher()
    {
        for (int i = 0 ; i < sommet ; i++) System.out.println( tab[i] ) ;
    }
    public boolean pileVide()
    {
        return sommet == 0 ;
    }

    public static void main(String args[]) {
        PileString p1 = new PileString(10) ;
        String t[] = {"Jesus", "Marie", "Joseph", "Le Diable"} ;
        // init de p1
        for (int i = 0 ; i < t.length ; i++) p1.empiler(t[i]) ;
        // on renverse p1 dans p2
        PileString p2 = new PileString(20) ;
        while(p1.pileVide() == false) p2.empiler(p1.depiler()) ;

        p1 = new PileString(20) ;
        // on renverse p2 dans p1
        while(p2.pileVide() == false) p1.empiler(p2.depiler()) ;

        p1.afficher() ;
    }
}
```

TABLE 2.4 – Manipulation de Piles

```
101 ANDREUX Jereme E
0 ANDRIAMIARINA Tsiorintsoa A
126 ANTCZAK Matthieu F
127 ANTONIAK Jereme F
```

TABLE 2.5 – Quelques lignes du fichier etudiant.txt

```
typedef struct etu {
  char* nom ;
  char* prenom ;
  char groupe[10] ;
} Etudiant1 ;
```

TABLE 2.6 – Définition du type Etudiant1

Chapitre 3

Les listes

3.1 Les listes chaînées

3.1.1 LinkedList

1. On souhaite traduire en `C` le programme Java donné Table 3.1 dont l'exécution donne le résultat suivant :

   ```
   jfmari@MacBook:C$ java LinkedListExample1
   Linked List Example!
   Linked list data: 44 33 22 11
   Linked list size: 4
   Linked list is empty
   jfmari@MacBook:C$
   ```

   Complétez le squelette du programme `C` donné Table 3.2 en remplaçant les points de suspension par les instruction `C` équivalentes. Utilisez la classe `Maillon` comme dans le projet Java.

2. Comparez les vitesses d'exécution de la version Java et de la version `C`.

3. On souhaite traduire en `C` le programme Java donné Table 3.3. Son exécution donne le résultat suivant :

   ```
   jfmari@macjfm1:C$ java LinkedListExample2 Janvier
   LinkedList contenait Janvier
   LinkedList ne contient pas maintenant Janvier
   jfmari@macjfm1:C$
   ```

   Complétez le squelette du programme `C` donné Table 3.4 en remplaçant les points de suspension par les instruction `C` équivalentes. Utilisez la classe `Maillon` comme dans le projet Java.

```java
import java.util.*;

public class LinkedListExample1{
    public static void main(String[] args) {
        System.out.println("Linked List Example!");
        LinkedList <Integer>list = new LinkedList<Integer>();
        int num1 = 11, num2 = 22, num3 = 33, num4 = 44;
        int listSize;
        Iterator iterator;
        //Adding data in the list
        list.addFirst(num1);
        list.addFirst(num2);
        list.addFirst(num3);
        list.addFirst(num4);
        listSize = list.size();
        System.out.print( "Linked list data: ");
        //Create a iterator
        iterator = list.iterator();
        while (iterator.hasNext()){
            System.out.print(iterator.next()+" ");
        }
        System.out.println();
        //Check list empty or not
        if (list.isEmpty()){
            System.out.println("Linked list is empty");
        }
        else{
            System.out.println( "Linked list size: " + listSize);
        }
        //Remove all
        list.clear();
        if (list.isEmpty()){
            System.out.println("Linked list is empty");
        }
        else{
            System.out.println( "Linked list size: " + list.size());
        }
    }
}
```

TABLE 3.1 – Manipulation d'une liste chaînée en Java

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct cellule_t {
  int val ;
  struct cellule_t *suc ;
} Maillon ;

Maillon* addFirst(Maillon* l, int v) {
  /* ajoute l'entier v a la liste de tete l */
  Maillon* p ;
  p = (Maillon*) malloc(sizeof(Maillon)) ;
...
  return p ;
}

int size(...)
{
  /* compte le nbr d'elements
  ** de la liste l dans n
  */
  int n = 0 ;
 ...
  return n ;
}

void clear(...)
{
  ...
}

int isEmpty(...)
{
...
}

int main()
{
  Maillon *list ;

  int num1 = 11, num2 = 22, num3 = 33, num4 = 44;
  int listSize;
  Maillon* p ;
  printf("Linked List Example!\n");
  /* Adding data in the list */
  list = addFirst(list, num1);
  list = addFirst(list, num2);
  list = addFirst(list, num3);
  list = addFirst(list, num4);
  listSize = size(list);
  printf( "Linked list data: ");
...
  return 0 ;
}
```

TABLE 3.2 – Manipulation de la même liste en C

```java
import java.util.LinkedList;

public class LinkedListExample2
{

    public static void main(String[] args)
    {

        LinkedList<String> lk_Liste = new LinkedList<String>();

        // Ajout des elements dans la liste
        String t [ ] = {"Fevrier", "Mars"} ; // init

        for (int i = 0 ; i < t.length ; i++) lk_Liste.addFirst(t[i]);
        lk_Liste.addLast("Decembre") ;
        // Verifier si args[0] existe dans le LinkedList
        if (lk_Liste.contains(args[0]))
            {
                System.out.println("LinkedList contenait " + args[0]);
            }
        else
            {
                System.out.println("LinkedList ne contenait pas " + args[0]);
            }
        // effacer la liste
        lk_Liste.clear() ;
        // Verifier a nouveau si args[0] existe dans le LinkedList
        if (lk_Liste.contains(args[0]))
            {
                System.out.println("LinkedList contient maintenant " + args[0]);
            }
        else
            {
                System.out.println("LinkedList ne contient pas maintenant " + args[0]);
            }
    }
}
```

TABLE 3.3 – Manipulation d'une liste chaînée en Java

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  /* maillon pour une liste chainee de chaines de caracteres */
6  typedef struct cellule_t {
7    char* str ;
8    struct cellule_t *suc ;
9  } Maillon ;
10
11 Maillon* addFirst(Maillon* l, char* v) {
12   /* ajoute la chaine v a l'attribut str de la liste de tete l */
13   Maillon* p ;
14   char *pc ;
15   p = (Maillon*) malloc(sizeof(Maillon)) ; /* nouveau maillon */
16   pc = malloc(strlen(v) + 1) ; /* place alloue'e pour ranger v */
17   ...
18   return p ; /* retourne la nouvelle tete de liste */
19 }
20
21 void  clear(...)
22 /* erase the list and free its elements */
23 {
24 ...
25 }
26
27 int contains(Maillon* l, char *v)
28 /* retourne 1 si v est dans la liste de tete l
29 ** sinon retourne 0 */
30 {
31 ...
32 }
33
34 Maillon* addLast(Maillon* l, char* v)
35 /* ajoute v a la fin de liste de tete l
36 ** retourne la tete de liste */
37 {
38 ...
39 }
40
41 int main(int argc, char **argv)
42 {
43   Maillon *list = NULL ;
44   char *t[2] = {"Fevrier", "Mars"} ; /* init */
45   int i ;
46
47   if (argc < 2) {
48     printf("Usage : %s chaine\n", argv[0]) ;
49     exit(1) ;
50   }
51
52   /* Ajout des elements dans la liste */
53   for (i = 0 ; i < 2 ; i++)
54     list = addFirst(list, t[i]) ;
55 ...
56   return 0 ;
```

3.1.2 La bibliothèque Liste d'entiers

On considère la liste formée d'éléments du type suivant :

```
typedef struct TCell {
 int val;
 struct TCell *suc;
 } Maillon ;
```

Une liste sera désignée par sa tête de type `Maillon*`.

- Compléter la bibliothèque `Liste` avec les fonctions suivantes :

  affichage des éléments : `void liste_afficher(Maillon*)` ;

  insertion d'un élément en tête de liste : `Maillon* tete_inserer(Maillon*, int)` ;

  insertion d'un élément en queue de liste : `Maillon* queue_inserer(Maillon*, int)` ;

  suppression d'un élément en queue de liste : `Maillon* queue_supprimer(Maillon*)` ;

  recherche d'un élément dans la liste : `int liste_rechercher(Maillon*, int)`. La fonction retournera le rang (compté à partir de 1) si l'élément est bien dans la liste. Sinon, la fonction retournera 0.

- Ecrire la fonction `int main()` qui permet de tester toutes ces fonctions. Comme dans les exercices prcédents, prévoir la présentation d'un menu regroupant toutes ces fonctions et permettant ainsi une interaction avec l'utilisateur.

3.1.3 Lecture de fichier texte et codes postaux

L'objectif de ce programme est de manipuler un fichier contenant tous les codes postaux des villes françaises. Ce fichier (`codePost.txt`) se présente sous la forme d'un fichier texte formaté en lignes comme suit :

```
aast      64460
abainville  55130
abancourt 59265
abancourt 60220
abaucourt-hautecourt      55400
```

1. Créer une classe (structure) `Ville`, correspondant à une ligne du fichier, avec deux attributs : le nom et le code postal de la ville (deux chaînes de caractères).

2. Ecrire un `int main(int argc, char **argv)` pour qui `argv[1]` est le nom du fichier de codes postaux. Ce programme lit le fichier et le stocke sous forme d'une liste chaînée dont le maillon sera :
```
typedef struct cellule_t {
  Ville* town ;
  struct cellule_t *suc ;
} Maillon ;
```
3. Définir une méthode `void afficher(int n, ...)` permettant d'afficher à l'écran ses n premiers éléments, ou tous ses éléments si n vaut 0. Cette méthode sera utiliséee pour valider les résultats obtenus par les différentes méthodes à définir ultérieurement.