V-2 (09-11-2012)

Pratique du C Structures autoréférentes

hardie de liste chaînée

V81 (09-11-2012)

Pratique du C Structures autoréférentes

V81 (09-11-2012)

# Pratique du C Structures autoréférentes

Licence Informatique — Université Lille 1 Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Semestre 4 — 2015-2016

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf V81 (09-11-2012)

Nous allons implanter les outils de manipulation d'une liste chaînée en nous imposant une contrainte forte : nous ne nous servirons pas des fonctions malloc et free.

Le seul espace mémoire disponible sera constitué d'un tableau de caractères :

```
#define SIZE 1<<16
char memoire[SIZE] ;
```

Ce type est uniquement choisi pour désigner des octets.

Nous devrons donc gérer l'allocation dynamique de la mémoire. Notez bien qu'il ne s'agit pas ici de donner des implantations rudimentaires des fonctions malloc et free.

En effet, dans notre cas, nous allouerons toujours la même taille de mémoire correspondant à une cellule.

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf V81 (09-11-2012)

# Les fonctions classiques de manipulation de ces listes (supposées triées)

```
int suisvide(cellule_t *liste){    return !liste ; }
void inserer( cellule_t *cell, cellule_t **liste ){
 cellule_t *tmp=NULL, *ptr = *liste ;
  while(ptr && ptr->contenu < cell->contenu){
   tmp = ptr ;
   ptr = ptr->next; }
  cell->next= ptr ;
 if(tmp) tmp->next= cell ;
  else *liste = cell ;
void extraire( cellule_t *cell, cellule_t **liste ){
 cellule_t *tmp=NULL, *ptr = *liste ;
  while(ptr && ptr != cell){
   tmp = ptr ;
   ptr = ptr->next ;
  if(tmp) tmp->next = cell ->next ;
  else *liste = NULL :
                                    www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf V81 (09-11-2012)
```

Pratique du C Structures autoréférentes

Manipulation hardie de liste chaînée

Pratique du C Structures autoréférentes

Manipulation hardie de liste chaînée

Définition d'objets auto-référents

L'idée de base est d'inclure dans la définition de l'objet des références à des objets de même type;

- ▶ arbres, listes, graphes, etc. : ce sont des nœuds référençant d'autres nœuds;
- ▶ ce sont des types récursifs. Par exemple, pour arbre binaire de recherche on a :

```
struct noeud {
  int value:
   struct noeud *gauche, *droit;
};
```

mais l'espace mémoire est réservé pour un pointeur sur une structure noeud; pas pour stocker un objet de

On peut définir des objets en référence croisée :

```
struct s {
                          struct t {
 struct t *p_t;
                             struct s *p_s;
                          }:
                                        www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf
```

# Les déclarations et définitions basiques des types et variables

```
#define NULL 0 /* pour d\'esigner la fin d'une liste */
struct cellule_m
 unsigned int contenu;
 struct cellule_m *next ;
typedef struct cellule_m cellule_t ;
/* pour compenser l'absence de fonction free, on utilise */
cellule_t *ListeDesCellulesLibres = NULL ;
/* on peut ensuite avoir autant de listes que n\'ecessaire */
cellule_t *maliste = NULL ;
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf

Les fonctions d'allocations de cellules

```
char *sommet = memoire :
void FreeCellule(cellule_t *cell){
 inserer( cell, &ListeDesCellulesLibres ) ;
cellule_t *MallocCellule(unsigned int stuff){
 cellule_t *tmp = ListeDesCellulesLibres ;
 if( suisvide(ListeDesCellulesLibres) ){
   extraire( tmp, &ListeDesCellulesLibres ) ;
   tmp->contenu = stuff ;
   tmp->next = NULL ;
   return tmp ;
 if(SIZE-(sommet-memoire)<sizeof(cellule t))</pre>
    return NULL ;
 tmp = (cellule_t *) sommet ;
 tmp->contenu = stuff ; tmp->next = NULL ;
 sommet = (char *) ((cellule_t *) sommet +1) ;
 return tmp;
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf

```
Pratique du C
Structures
```

Principes généraux

hardie de liste chaînée

Parcours de labyrinthe par pi

Arbre binaire de

recherche

V81 (09-11-2012)

Pratique du C Structures autoréférentes

généraux

Manipulation hardie de liste

Parcours de labyrinthe par pile

Arbre binaire de recherche

V81 (09-11-2012)

Pratique du C
Structures
autoréférentes

Principes généraux Manipulation hardie de liste

Arbre binaire de

V81 (09-11-2012)

On se propose de parcourir un labyrinthe que l'on aura préalablement représenté par un tableau.

```
#define FERME 0
#define OUVERT 1
#define PARCOURU 2
#define LONGUEUR 5
#define LARGEUR 6
char petitlab[LONGUEUR][LARGEUR] = {
  {FERME, OUVERT, FERME, FERME, FERME},
  {FERME, OUVERT, OUVERT, OUVERT, FERME},
  {FERME, OUVERT, FERME, FERME, FERME},
  {FERME, OUVERT, OUVERT, OUVERT, FERME},
  {FERME, FERME, FERME, OUVERT, FERME}};
XXXXX
   Х
X X X
XXX
X X On commence et on termine toujours aux m\^emes endroits
XXXXX */
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf V81 (09-11-2012)

Il nous faut maintenant implanter les fonctions classiques de manipulations de pile :

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf V81 (09-11-2012)

#### **Fonctions auxiliaires**

Pour gérer les chemins **inefficacement**, on utilise la fonction char \* concat(char \*, char) qui prend en entrée un chemin (disons "ws"), un déplacement ('s') et retourne un pointeur sur la nouvelle solution ainsi formée ("wss") pour laquelle de la mémoire aura été réservée (l'ancien chemin n'est pas modifié). Ceci modifie l'empilement et le dépilement :

```
void empiler(Pas_t step, Pile_t *pile, char *chemin){
    cell_t *tmp = (cell_t *) malloc (sizeof(cell_t));
    tmp->chemin = chemin;
    tmp->pas = step;
    tmp->next = *pile;
    *pile = tmp; }

Pas_t depiler(Pile_t *pile){
    cell_t *tmp = *pile;
    Pas_t res = tmp->pas;
    *pile = tmp->next;
    free(tmp->chemin);
    free(tmp); return res; }

*www.fil.univ-lilel.fr/*sedoglav/C/Cours07.pdf V81 (09-11-2012)
```

Pratique du C Structures

Principes

Manipulation hardie de liste

Parcours de labyrinthe par pile

Arbre binaire de

Pratique du C Structures autoréférentes

Parcours de labyrinthe par pile

Pratique du C Structures autoréférentes

Parcours de

labyrinthe par pile

Implantation d'une pile Pour ce faire, nous utilisons une

Pour ce faire, nous utilisons une pile contenant des *pas* et implanter par une liste (sans s'occuper de la base mais uniquement du sommet) :

```
struct Pas
{
  int x;
  int y;
};
typedef struct Pas Pas_t;

struct cell
{
  struct cell * next;
  Pas_t pas;
  char *chemin; /* nous verrons plus tard */
};  /* \'a quoi \c{c}a sert */

typedef struct cell cell_t;

typedef cell_t * Pile_t;
```

# Algorithme pour savoir si on peut sortir

Pour parcourir notre labyrinthe, nous allons utiliser une variable globale représentant notre pile et qui contiendra les pas à explorer. Au début du parcours, elle ne contient que le seul pas allant de (0,1) à (1,1).

Tant que notre pile n'est pas vide, on retire le pas de tête. S'il est possible, on réaffiche le labyrinthe en prenant soin de marquer d'un signe les cases déjà visitées pour ne pas boucler. Puis on ajoute à la pile tous les pas possibles (pour lesquels la case d'arrivée est franchissable) à partir du point d'arrivée du pas considéré.

Il suffit d'itérer ce qui précède pour sortir du labyrinthe. De plus, si on stocke à chaque pas la direction prise, on peut mémoriser la sortie.

Par exemple, on peut utiliser une chaîne de caractères initialement vide à laquelle, on peut ajouter les lettres : 'n', 'e', 's', 'w' pour signifier nord, etc. (par convention, on ne se déplace pas en diagonale).

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf

### Le début de la fonction principale

```
#include<stdio.h>
#include<stdiib.h>
#include <string.h>
#include"Labyrinthe.h"
#include"LabyrintheExemple.h"

extern void empiler(Pas_t, Pile_t *, char *);
extern Pas_t depiler(Pile_t *);
extern int estVide(Pile_t *);

char * concat(char *,char);
void affichage(char lab[LONGUEUR][LARGEUR]);
char * parcourir(Pile_t *, char lab[LONGUEUR][LARGEUR]);

void mystrcpy(char **, char *);
/* exercice": donner la d\'efinition de cette fonction
    adapt\'ee \'a la situation d\'ecrite dans les
    transparents suivants */
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf

Parcours de labyrinthe par pile

V81 (09-11-2012)

Pratique du C Structures autoréférentes

triés

Arbre binaire de

recherche

V81 (09-11-2012)

Pratique du C Structures autoréférentes

Arbre binaire de recherche

V81 (09-11-2012)

# main

La fonction principale

```
int
(void)
  char *path = "" ;
  /* Le premier pas */
                                                                     Parcours de 
labyrinthe par pile
  Pas_t PointDeDepart = {.x = 0 ,.y = 1 };
  Pile_t mapile = NULL ;
  empiler(PointDeDepart, &mapile, path) ;
  /* On considere que l'entr\'ee est parcouru */
  petitlab[PointDeDepart.x] [PointDeDepart.y] = PARCOURU ;
  /* Parcours et analyse du resultat */
 path = parcourir(&mapile,petitlab) ;
  if( path )
   printf("\n Felicitation,
             vous \\^etes sorti par le chemin %s\n",path);
   printf("\n ;-( ce labyrinthe n'a pas de sortie\n");
 return 0 ;
                                      www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf V81 (09-11-2012)
```

# Un programme qui affiche ses arguments

L'option -r indique que l'ordre est décroissant.

```
Le fichier abr.h:
 struct noeud
                                                                 Arbre binaire de
    int v ;
   struct noeud *fg, *fd;
   typedef struct noeud Noeud ;
   typedef Noeud * Abr ;
   void init(Abr *):
   void inserer(Abr *, int);
   void imprimer_croissant(Abr);
   void imprimer_decroissant(Abr);
```

#### www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf V81 (09-11-2012)

# Suite du fichier abr.c

```
Pratique du C
Structures
autoréférentes
#include <stdio.h>
void
imprimer_croissant
(Abr a)
   if (a) {
       imprimer_croissant (a->fg) ;
       printf ("%d\n", a->v);
                                                                            Arbre binaire de
       imprimer_croissant (a->fd) ;
}
imprimer_decroissant
(Abr a)
       imprimer_decroissant (a->fd) ;
       printf ("%d\n", a->v);
       imprimer_decroissant (a->fg) ;
   }
}
                                          www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf V81 (09-11-2012)
```

### La fonction de parcours

```
char * parcourir(Pile_t *pile, char lab[LONGUEUR][LARGEUR]){
  Pas_t pos,nextpos, pas[4] = { \{0,1\},\{1,0\},\{0,-1\},\{-1,0\}\} ;
  char dep[4] = { 's', 'e',
                                'n', 'w'};
  char *path = NULL; int i ;
  do{ mystrcpy(&path,(*pile)->chemin) ;
    pos = depiler(pile) ;
    if (pos.x==LONGUEUR-1 && pos.y==LARGEUR-2)
      return path ; /* C'est gagn\'e on est sorti */
    for(i=0; i<4; i++){
     nextpos.x = pos.x+pas[i].x ; /* On prepare les pas */
      nextpos.y = pos.y+pas[i].y ; /* \'a empiler
      if (lab[nextpos.x][nextpos.y] == OUVERT) {
        lab[nextpos.x][nextpos.y] = PARCOURU;
        empiler(nextpos,pile,concat(path,dep[i])) ;
        affichage(lab);
       printf("\n") ;
   }
  } while (estVide(pile)) ;
  return 0 ; }
                                    www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf
```

### Le fichier abr.c:

Pratique du C Structures autoréférentes

```
#include <stdlib.h>
 void
 init
 (Abr *a)
    *a = NULL ;
void
inserer
(Abr *a, int v)
 if (! *a) {
     *a = (Abr) malloc (sizeof (struct noeud));
     (*a)->v = v ;
     (*a)->fg = (*a)->fd = NULL ;
  } else if (v <= (*a)->v)
     inserer (& (*a)->fg, v);
    inserer (& (*a)->fd, v) ;
                                   www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf
```

## Le fichier main.c

```
int
main
(int argc, char *argv[])
  char order = 0;
  Abr a;
  if (argc < 2) return 1;
  if (!(strcmp(argv[1], "-r")))
    order=1;
    argc-=1; argv+=1;
  init(&a);
  while (--argc) inserer(&a, atoi(*++argv));
  if (order)
   imprimer_decroissant(a);
   imprimer_croissant(a);
  return 0 ;
                                  www.fil.univ-lille1.fr/~sedoglav/C/Cours07.pdf
```

```
Pratique du C
Structures
autoréférentes
```

Manipulation hardie de liste

Parcours de

Arbre binaire de

V81 (09-11-2012)

### Arbre binaire : insertion itérative

Insertion itérative dans l'arbre

```
#define allouer (struct noeud *) malloc(sizeof(struct noeud))
void inserer_iter(Abr *a, int v) {
enum {doite, gauche} direction;
Abr pere = NULL, current = *a; while (current) {
 pere = current;
  if (v <= current->v) {
   dir = gauche;
   current = current->fg;
 } else {
   dir = droite;
   current = currrent->fd;
  if (pere)
   if (dir == gauche) current = pere->fg = allouer ;
   else current = pere->fd = allouer ;
  else current = *a = allouer ;
  current->v = v;
  current->fg = current->fd = NULL;
```