ISIMA Première Année

Compte-Rendu de TP

Structures de Données

Dérécursification à l'aide d'une pile

Benjamin BARBESANGE
Pierre-Loup PISSAVY
Groupe G21

Enseignant :
Michelle CHABROL

février 2015

ISIMA

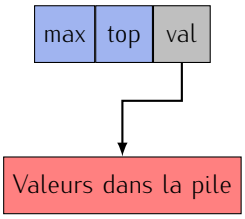
1 | Présentation

Le but de ce TP est de dérécursifier une fonction à l'aide d'un pile. Les fonctions de gestion d'une pile seront ainsi crées. Les opérations suivantes sont permisesavec la pile :

- Initialiser la pile,
- Libérer la pile,
- Tester si la pile est vide,
- Tester si la pile est pleine,
- Retourner l'élément en haut de la pile,
- Afficher l'élément en haut de la pile,
- Insérer un élément dans la pile.

1.1 Structure de données employée

(a) Structure utilisée



(b) Code

Code C

```
17 typedef struct _stack_t {
18   int       max;
19   int       top;
20   datatype *val;
21 } stack_t;
```

FIGURE 1.1 – Structure et code correspondant

1.2 Organisation du code source

Nous avons défini deux modules, le premier contient une fonction sous forme récurssive ansi que sa version sous forme itérative. Nous disposons également d'un module permettant de gérer une pile, qui est ainsi utilisée lors de la dérécursification de la fonction.

1.2.1 Gestion de la pile

- `src/stack.h`
- `src/stack.c`

1.2.2 Fonction récursive

- `src/truc.h`
- `src/truc.c`

1.2.3 Programme principal

- `src/main.c`

2 | Détails du programme

2.1 Gestion de la pile

Code C

```c
/*  stack.h
  Header

  -----| DERECURSIFICATION DE FONCTION PAR PILE |-----

  BARBESANGE Benjamin,
  PISSAVY Pierre-Loup

  ISIMA 1ere Annee, 2014-2015
*/

#ifndef __STACK__H
#define __STACK__H

  typedef int datatype;

  typedef struct _stack_t {
    int       max;
    int       top;
    datatype *val;
  } stack_t;

  int init(stack_t *,int);
  void supp(stack_t *);
  int empty(stack_t);
  int full(stack_t);
  int pop(stack_t *, datatype *);
  int top(stack_t *, datatype *);
  int push(stack_t *, datatype);

#endif
```

Code C

```c
/*  stack.c
  Fonctions de gestion de la structure de pile

  -----| DERECURSIFICATION DE FONCTION PAR PILE |-----

  BARBESANGE Benjamin,
  PISSAVY Pierre-Loup

  ISIMA 1ere Annee, 2014-2015
*/

#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

int init(stack_t *p, int n) {
  int ret = 1;
  p->max = n;
  p->top = -1;
  p->val = (int*) malloc(n*sizeof(int));
  if (p->val == NULL) {
    ret = 0;
  }
  return ret;
}

void supp(stack_t *p) {
  free(p->val);
}

int empty(stack_t p) {
  return (p.top == -1)?1:0;
}

int full(stack_t p) {
  return (p.top == p.max-1)?1:0;
}

int pop(stack_t *p, datatype *v) {
  int ok = 0;
  if (!empty(*p)) {
    *v = p->val[p->top];
    ok = 1;
    p->top--;
  }
  return ok;
}

int top(stack_t *p, datatype *v) {
  int ok = 0;
  if (!empty(*p)) {
    *v = p->val[p->top];
    ok = 1;
  }
  return ok;
```

```c
}

int push(stack_t *p, datatype v) {
  int ok = 0;
  if (!full(*p)) {
    p->top++;
    p->val[p->top] = v;
    ok = 1;
  }
  return ok;
}
```

2.2 Fonction récursive

Code C

```c
/*  truc.h
  Header

  -----| DERECURSIFICATION DE FONCTION PAR PILE |-----

  BARBESANGE Benjamin,
  PISSAVY Pierre-Loup

  ISIMA 1ere Annee, 2014-2015
*/

#ifndef __TRUC__H
#define __TRUC__H

/* Fonction sous forme recurssive */
int TRUC(int, int);

/* Variante iterative de la fonction */
int truc_iter(int, int);

#endif
```

Code C

```c
/*  truc.c
  Fonction recursive et son equivalent en iteratif

  -----| DERECURSIFICATION DE FONCTION PAR PILE |-----

  BARBESANGE Benjamin,
  PISSAVY Pierre-Loup

  ISIMA 1ere Annee, 2014-2015
*/

#include <stdio.h>
#include "stack.h"
#include "truc.h"

#define N 10
```

```c

int P[N+1] = {0,1,3,2,5,5,2,7,1,9,1};

/*  int TRUC(int S, int I)
  Fonction sous forme recursive qui affiche la decomposition de
  S en I entiers pris a partir d'un tableau d'entiers (defini ici en statique)

  Entrees :
    int S : Nombre a decomposer
    int I : Nombre d'entiers utilises pour decomposer S

  Sortie :
    int : entier sous forme de booleen
      0 si on a pas pu decomposer S exactement
      1 sinon
*/
int TRUC(int S, int I) {
  if (S == 0) {
    return 1;
  } else if (S < 0 || I > N) {
    return 0;
  } else if (TRUC(S-P[I],I+1)) {
    printf("%d\n",P[I]);
    return 1;
  } else {
    return TRUC(S,I+1);
  }
}

/*  int truc_iter(int s, int i)
  Meme fonction qu'au dessus, mais sous forme iterative

  Entrees :
    int s : Nombre a decomposer
    int i : Nombre d'entiers utilises pour decomposer S

  Sortie :
    int : entier sous forme de booleen
      0 si on a pas pu decomposer S exactement
      1 sinon
*/
int truc_iter(int s, int i) {
  int sl = s;
  int il = i;
  int r;
  stack_t p;
  init(&p,N);
  do {
    while (sl > 0 && il <= N) {
      push(&p,sl);
      push(&p,il);
      sl -= P[il];
      ++il;
    }
    if (sl == 0) {
      r = 1;
```

```
      while (!empty(p)) {
        pop(&p,&il);
        pop(&p,&sl);
        printf("%d\n",P[il]);
      }
    } else {
      r = 0;
      if (!empty(p)) {
        pop(&p,&il);
        pop(&p,&sl);
        ++il;
      }
    }
  } while (!empty(p));
  return r;
}
```

2.3 Programme principal

Code C

```c
#include <stdio.h>
#include "stack.h"

int main(void) {
  stack_t p;
  int i = 0;
  if (init(&p,10)) {
    while (push(&p,i)) {
      printf("Empiler: %d\n",i);
      ++i;
    }
    while (pop(&p,&i)) {
      printf("Depiler: %d\n",i);
    }
    supp(&p);
  }
  return 0;
}
```

3 | Principes et lexiques des fonctions

Dans cette partie, sont décrits les algorithmes de principe associés aux fonctions écrites en langage C, ainsi qu'un lexique concernant les variables intermédiaires des fonctions.

Le lexique des variable d'entrée, sortie et entrée/sortie sont disponibles dans le code source directement.

3.1 Gestion de la pile

La gestion de la pile s'effectue grace aux fichiers *stack.c* et *stack.h*.

3.1.1 init

3.1.2 supp

3.1.3 empty

3.1.4 full

3.1.5 pop

3.1.6 top

3.1.7 push

3.2 Dérecursification de la fonction

La fonction récursive ainsi que sa version itérée se trouvent dans les fichiers *truc.c* et *truc.h*.

3.2.1 TRUC

Cette fonction étant l'énoncé du TP, nous ne détaillerons ainsi ne le principe ni les variables utilisées dans cet algorihtme.

3.2.2 truc_iter

```
Algorithme truc_iter (Principe)
Début
    Copie des paramètres d'entré dans des variables locales, sl et il;
    Initialisation de la pile de la même taille que le tableau statique;
    Répéter
        TantQue sl > 0 Et Alors  il <= N Faire
            On push sl dans la pile;
            On push il dans la pile;
            sl = sl - P[il];
            On incrémente il;
        FinTantQue;
        Si sl = 0 Alors
            Le booléen de retour est à Vrai;
            TantQue la pile n'est pas vide Faire
                On récupère il et sl à partir de la pile;
                On affiche P[il];
            FinTantQue;
        Sinon
            Le booléen de retour est à Faux;
            Si la pile n'est pas vide Alors
                On récupère il est sl à partir de la pile;
                On incrémente il;
            FinSi;
        FinSi;
    TantQue la pile n'est pas vide fait;
    Retourner Booléen de retour;
Fin
```

Lexique :

sl : *copie locale du nombre s passé en paramètre. Représente le nombre à décomposer*

il : *copie locale du nombre i passé en paramètre. Représente le nombre d'entiers du tableau à utiliser pour décomposer s*

r : *booléen de retour, indique 1 si on a obtenue la somme s, 0 sinon*

p : *pile*

P : *tableau d'entiers, défini statiquement*

N : *taille du tableau P*

4 | Compte rendu d'exécution

4.1 Makefile

Makefile

```
#Compilateur et options de compilation
CC=gcc
CFLAGS=-Wall -ansi -pedantic -Wextra -g

#Fichiers du projet
SOURCES1=main.c stack.c
SOURCES2=truc.c stack.c
OBJECTS1=$(SOURCES1:.c=.o)
OBJECTS2=$(SOURCES2:.c=.o)

#Nom du programme
EXEC1=prog
EXEC2=truc

all: $(EXEC1) $(EXEC2)


$(EXEC1): $(OBJECTS1)
	$(CC) $(CFLAGS) $^ -o $(EXEC1)

$(EXEC2): $(OBJECTS2)
	$(CC) $(CFLAGS) $^ -o $(EXEC2)

.c.o:
	$(CC) -c $(CFLAGS) $*.c

clean:
	rm $(OBJECTS1) $(EXEC1) $(EXEC2) truc.o
```

4.2 Jeux de tests

Exécution du programme avec le fichier suivant :