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

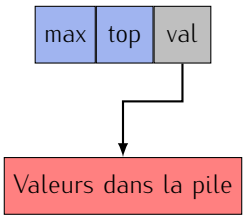
1 | Présentation

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

0.5 Structure utilisée



0.5 Code

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

-
-

1.2.2 Fonction récursive

-
-

1.2.3 Programme principal

-

2 | Détails du programme

2.1 Gestion de la pile

2.2 Fonction récursive

2.3 Programme principal

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

```
Algorithme init
Début
    On initialise le code de retour à 0;   [ Il n'y a aucune erreur ]
    On initialise la taille max de la pile;
    On initialise l'indice du haut de la pile à -1;   [ Pour indiquer qu'elle est vide ]
    On alloue l'espace de la pile;
    Si l'allocation est réussie Alors
        Le code d'erreur passe à 0;
    FinSi;
    Retourner code d'erreur;
Fin
```

Lexique :

ret : *code d'erreur, 0 si il y a une erreur de création de la pile, 1 sinon*

3.1.2 supp

Ici, nous libérons simplement de tableau de valeurs de la pile, puisque celui-ci est alloué dynamiquement lors de la création.

3.1.3 empty

Cette fonction teste simplement si l'indice du haut de la pile est -1, ce qui veut dire qu'il n'y a aucun élément dans la pile. Ainsi la valeur 1 sera retournée. Sinon la valeur 0 est retournée.

3.1.4 full

Cette fonction vérifie si l'indice de l'élément en haut de la pile est égal à la taille max de la pile (moins 1, car les tableaux commencent à 0). Si c'est le cas, on renvoie 1 pour signaler que la pile est pleine, et 0 sinon.

3.1.5 pop

```
Algorithme pop
Début
    On initialise le code de retour à 0;   [ On n'a pas dépilé ]
    Si la pile n'est pas vide Alors
        On dépile l'élément dans une variable en Input/Output;
        Le code d'erreur passe à 1;   [ On a dépilé et récupéré l'élément ]
        On modifie l'indice de l'élément en haut de la pile;   [ On retranche 1 à l'indice précédent ]
    FinSi;
    Retourner code d'erreur;
Fin
```

Lexique :

ok : *code d'erreur, 0 si on n'a pas dépilé, 1 si on a dépilé la valeur en haut de la pile*

3.1.6 top

```
Algorithme top
Début
    On initialise le code de retour à 0;   [ Pas d'élément dépilé ]
    Si la pile n'est pas vide Alors
        On dépile dans une variable en I/O;
        Le code d'erreur passe à 1;   [ On a dépilé ]
    FinSi;
    Retourner code d'erreur;
Fin
```

Lexique :

ok : *code d'erreur, 0 si on n'a pas récupéré l'élément en haut de la pile, 1 si on l'a récupéré*

3.1.7 push

Algorithme push
Début
    On initialise le code de retour à 0; *[L'élément n'est pas ajouté dans la pile]*
    Si la pile n'est pas pleine Alors
        On incrémente l'indice de l'élément en haut de la pile;
        On place l'élément dans le tableau de la pile, à l'indice précédement modifié;
        Le code d'erreur passe à 1; *[L'élément est empilé]*
    FinSi;
    Retourner code d'erreur;
Fin

Lexique :

ok : *code d'erreur, 0 si on n'a pas empilé, 1 si on a empilé la valeur*

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

4.2 Jeux de tests

Exécution du programme avec le fichier suivant :