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

mars 2015

ISIMA

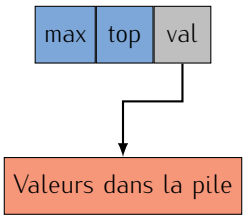
1 | Présentation

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

Code C

```
17
18  ktypedef k+ktint ndatatypep; c+cm/* permet dutiliser des types differents avec la pile */
19
20  ktypedef kstruct nstackt p
21    k+ktint       nmaxp;  c+cm/* nombre max delements dans la pile */
```

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+cm/*  stack.h
c+cm  Header

c+cm  -----| DERECURSIFICATION DE FONCTION PAR PILE |-----

c+cm  BARBESANGE Benjamin,
c+cm  PISSAVY Pierre-Loup

c+cm  ISIMA 1ere Annee, 2014-2015
c+cm*/

c+cpifndef STACKH
c+cpdefine STACKH

c+cp  include <stdio.h>
c+cp  include <stdlib.h>

  ktypedef k+ktint ndatatypep; c+cm/* permet dutiliser des types differents avec la pile */

  ktypedef kstruct nstackt p
    k+ktint       nmaxp;  c+cm/* nombre max delements dans la pile */
    k+ktint       ntopp;  c+cm/* position de lelement en tete de pile */
    ndatatype o*nvalp;  c+cm/* tableau des valeurs de la pile */
  p nstacktp;

  k+ktint ninitp(nstackt o*p,k+ktintp);
  k+ktvoid nsuppp(nstackt o*p);
  k+ktint nemptyp(nstacktp);
  k+ktint nfullp(nstacktp);
  k+ktint npopp(nstackt o*p, ndatatype o*p);
  k+ktint ntopp(nstackt o*p, ndatatype o*p);
  k+ktint npushp(nstackt o*p, ndatatypep);

c+cpendif
```

Code C

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

c+cm  -----| DERECURSIFICATION DE FONCTION PAR PILE |-----

c+cm  BARBESANGE Benjamin,
c+cm  PISSAVY Pierre-Loup

c+cm  ISIMA 1ere Annee, 2014-2015
c+cm*/

c+cpinclude "stack.h"

c+cm/*  int init(stackt *p, int n)
c+cmFonction dinitialisation de la pile avec une taille max

c+cm  Entrees :
c+cm    *p : pointeur sur la pile
c+cm    n : taille maximum de la pile

c+cm  Sortie :
c+cm    int : code derreur
c+cm      0 si aucune erreur
c+cm      1 si erreur de creation de la pile
c+cm*/
k+ktint n+nfinitp(nstackt o*npp, k+ktint nnp) p
  k+ktint nret o= l+m+mi1p;
  npo->nmax o= nnp;
  npo->ntop o= o-l+m+mi1p;
  npo->nval o= p(k+ktinto*p) nmallocp(nno*ksizeofp(k+ktintp));
  kif p(npo->nval o== n+nbNULLp) p
    nret o= l+m+mi0p;
  p
  kreturn nretp;
p

c+cm/*  void supp(stackt *p)
c+cmFonction de suppression de la pile

c+cm  Entree :
c+cm    *p : pointeur sur la tete de la pile

c+cm  Sortie :
c+cm    Aucune
c+cm*/
k+ktvoid n+nfsuppp(nstackt o*npp) p
  nfreep(npo->nvalp);
p

c+cm/*  int empty(stackt *p)
c+cmTeste si la pile est vide ou non

c+cm  Entree :
c+cm    p : tete de la pile

```

```
c+cm  Sortie :
c+cm    int : booleen
c+cm      0 si la pile nest pas vide
c+cm      1 si la pile est vide
c+cm*/
k+ktint n+nfemptyp(nstackt npp) p
  kreturn p(npp.ntop o== o-l+m+mi1p)o?l+m+mi1o:l+m+mi0p;
p

c+cm/*  int full(stackt p)
c+cmTeste si la pile est pleine ou non

c+cm  Entree :
c+cm    p : tete de la pile

c+cm  Sortie :
c+cm    int : booleen
c+cm      0 si la pile nest pas pleine
c+cm      1 si la pile est pleine
c+cm*/
k+ktint n+nffullp(nstackt npp) p
  kreturn p(npp.ntop o== npp.nmaxo-l+m+mi1p)o?l+m+mi1o:l+m+mi0p;
p

c+cm/*  int pop(stackt *p, datatype *v)
c+cmRecupere le premier element de la pile (et lenleve) et retourne un code derreur

c+cm  Entree :
c+cm    *p : pointeur sur la tete de la pile
c+cm    *v : pointeur sur un element du type de la pile, variable en I/O

c+cm  Sortie :
c+cm    int : code derreur
c+cm      0 si rien nest retourne dans la variable v
c+cm      1 si on a recupere lelement en tete
c+cm*/
k+ktint n+nfpopp(nstackt o*npp, ndatatype o*nvp) p
  k+ktint nok o= l+m+mi0p;
  kif p(o!nemptyp(o*npp)) p
    o*nv o= npo->nvalp[npo->ntopp];
    nok o= l+m+mi1p;
    npo->ntopo--p;
  p
  kreturn nokp;
p

c+cm/*  int top(stackt *p, datatype *v)
c+cmRetourne lelement en tete de la pile (sans lenlever) et retourne un code derreur

c+cm  Entree :
c+cm    *p : pointeur sur la tete de la pile
c+cm    *v : pointeur sur un element du type de la pile, variable en I/O

c+cm  Sortie :
c+cm    int : code derreur
c+cm      0 si rien nest retourne
```

```
c+cm      1 si on recupere lelement en tete
c+cm*/
k+ktint n+nftopp(nstackt o*npp, ndatatype o*nvp) p
  k+ktint nok o= l+m+mi0p;
  kif p(o!nemptyp(o*npp)) p
    o*nv o= npo->nvalp[npo->ntopp];
    nok o= l+m+mi1p;
  p
  kreturn nokp;
p

c+cm/*  int push(stackt *p, datatype v)
c+cmInsere un element en tete de la pile

c+cm  Entree :
c+cm    *p : pointeur sur la tete de la pile
c+cm    v : element a inserer dans la pile

c+cm  Sortie :
c+cm    int : code derreur
c+cm      0 si lelement nest pas ajoute dans la pile
c+cm      1 si lelement est ajoute dans la pile
c+cm*/
k+ktint n+nfpushp(nstackt o*npp, ndatatype nvp) p
  k+ktint nok o= l+m+mi0p;
  kif p(o!nfullp(o*npp)) p
    npo->ntopo++p;
    npo->nvalp[npo->ntopp] o= nvp;
    nok o= l+m+mi1p;
  p
  kreturn nokp;
p
```

2.2 Fonction récursive

Code C

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

c+cm  -----| DERECURSIFICATION DE FONCTION PAR PILE |-----

c+cm  BARBESANGE Benjamin,
c+cm  PISSAVY Pierre-Loup

c+cm  ISIMA 1ere Annee, 2014-2015
c+cm*/

c+cpinclude "truc.h"

c+cpdefine N 10

k+ktint nPp[nNo+l+m+mi1p] o= pl+m+mi0p,l+m+mi1p,l+m+mi3p,l+m+mi2p,l+m+mi0p,l+m+mi5p,l+m+mi2p,l+m+mi7p,l+m+m


```

```
c+cm/*  int TRUC(int S, int I)
c+cm  Fonction sous forme recursive qui affiche la decomposition de
c+cm  S en I entiers pris a partir dun tableau dentiers (defini ici en statique)

c+cm  Entrees :
c+cm    int S : Nombre a decomposer
c+cm    int I : Nombre dentiers utilises pour decomposer S

c+cm  Sortie :
c+cm    int : entier sous forme de booleen
c+cm      0 si on a pas pu decomposer S exactement
c+cm      1 sinon
c+cm*/
k+ktint n+nfTRUCp(k+ktint nSp, k+ktint nIp) p
  kif p(nS o== l+m+mi0p) p
    kreturn l+m+mi1p;
  p kelse kif p(nS o< l+m+mi0 o|| nI o> nNp) p
    kreturn l+m+mi0p;
  p kelse kif p(nTRUCp(nSo-nPp[nIp],nIo+l+m+mi1p)) p
    nprintfp(l+s"dl+s+senl+s"p,nPp[nIp]);
    kreturn l+m+mi1p;
  p kelse p
    kreturn nTRUCp(nSp,nIo+l+m+mi1p);
  p
p

c+cm/*  int truciter(int s, int i)
c+cm  Meme fonction quau dessus, mais sous forme iterative

c+cm  Entrees :
c+cm    int s : Nombre a decomposer
c+cm    int i : Nombre dentiers utilises pour decomposer S

c+cm  Sortie :
c+cm    int : entier sous forme de booleen
c+cm      0 si on a pas pu decomposer S exactement
c+cm      1 sinon
c+cm*/
k+ktint n+nftruciterp(k+ktint nsp, k+ktint nip) p
  k+ktint nsl o= nsp;
  k+ktint nil o= nip;
  k+ktint nr o= l+m+mi0p;
  nstackt npp;
  kif p(ninitp(o&npp,nNp)) p
    kdo p
      kwhile p(nsl o> l+m+mi0 o&& nil o<= nNp) p
        npushp(o&npp,nilp);
        nsl o-= nPp[nilp];
        o++nilp;
      p
      kif p(nsl o== l+m+mi0p) p
        nr o= l+m+mi1p;
        kwhile p(o!nemptyp(npp)) p
          npopp(o&npp,o&nilp);
          nsl o+= nPp[nilp];
          nprintfp(l+s"dl+s+senl+s"p,nPp[nilp]);
```

```
        p
      p kelse p
        nr o= l+m+mi0p;
        kif p(o!nemptyp(npp)) p
          npopp(o&npp,o&nilp);
          nsl o+= nPp[nilp];
          o++nilp;
        p
      p
    p kwhile p(o!nemptyp(npp));
    nsuppp(o&npp);
  p
  kreturn nrp;
p
```

2.3 Programme principal

Code C

```
c+cm/*  main.c
c+cm  Fonction principale du programme, pour les tests

c+cm  -----| DERECURSIFICATION DE FONCTION PAR PILE |-----

c+cm  BARBESANGE Benjamin,
c+cm  PISSAVY Pierre-Loup

c+cm  ISIMA 1ere Annee, 2014-2015
c+cm*/

c+cpinclude <stdio.h>
c+cpinclude <stdlib.h>
c+cpinclude "truc.h"
c+cpinclude "stack.h"

k+ktint n+nfmainp(k+ktint nargcp, k+ktchar o*nargvp[]) p
  nstackt npp;
  k+ktint ni o= l+m+mi0p;
  kif p(ninitp(o&npp,l+m+mi10p)) p
    kwhile p(npushp(o&npp,nip)) p
      nprintfp(l+s"Empiler: dl+s+senl+s"p,nip);
      o++nip;
    p
    kwhile p(npopp(o&npp,o&nip)) p
      nprintfp(l+s"Depiler: dl+s+senl+s"p,nip);
    p
    nsuppp(o&npp);
  p
  kif p(nargc o> l+m+mi2p) p
    nTRUCp(natoip(nargvp[l+m+mi1p]),natoip(nargvp[l+m+mi2p]));
    ntruciterp(natoip(nargvp[l+m+mi1p]),natoip(nargvp[l+m+mi2p]));
  p
  kreturn l+m+mi0p;
p
```

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

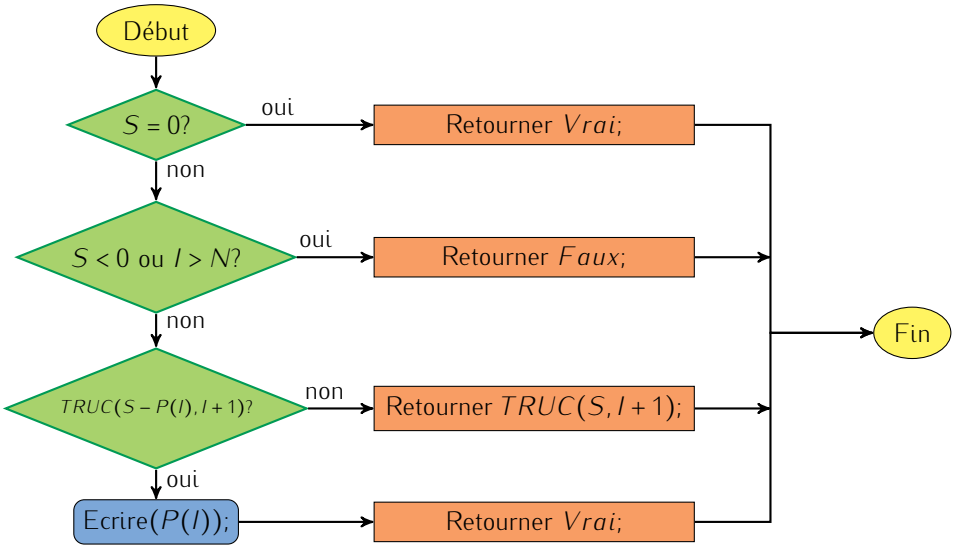
La fonction récursive ainsi que sa version itérée se trouvent dans les fichiers *truc.c* et *truc.h*.

3.2.1 TRUC

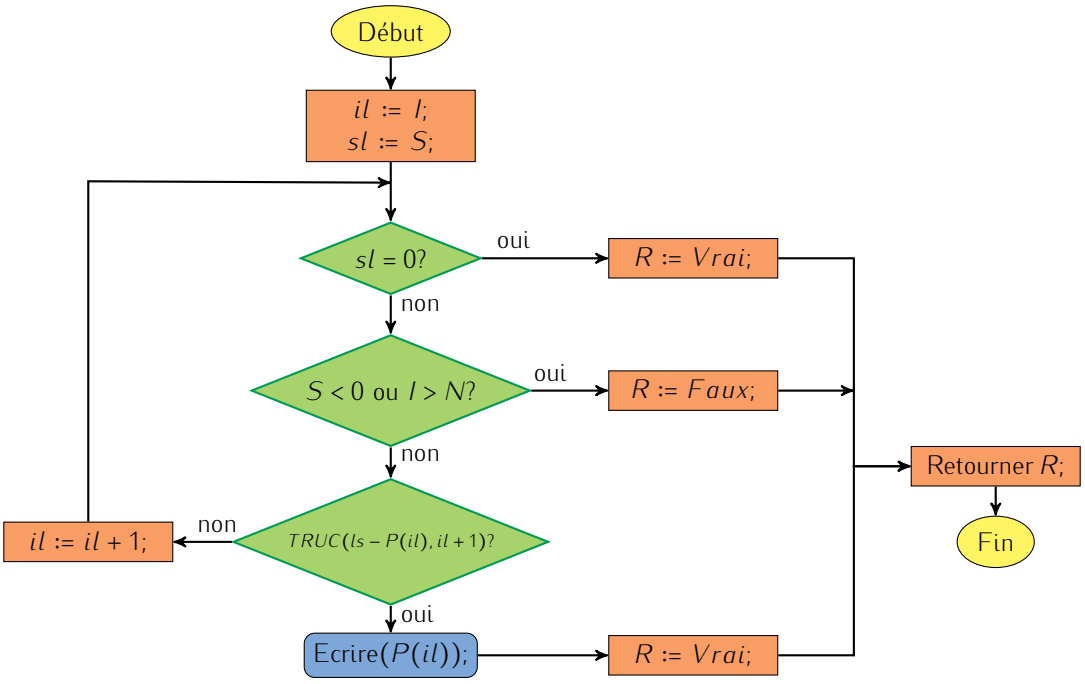
Cette fonction étant l'énoncé du TP, nous ne détaillerons ainsi ne le principe ni les variables utilisées dans cet algorithme.

Nous allons dérécursiver cette fonction.

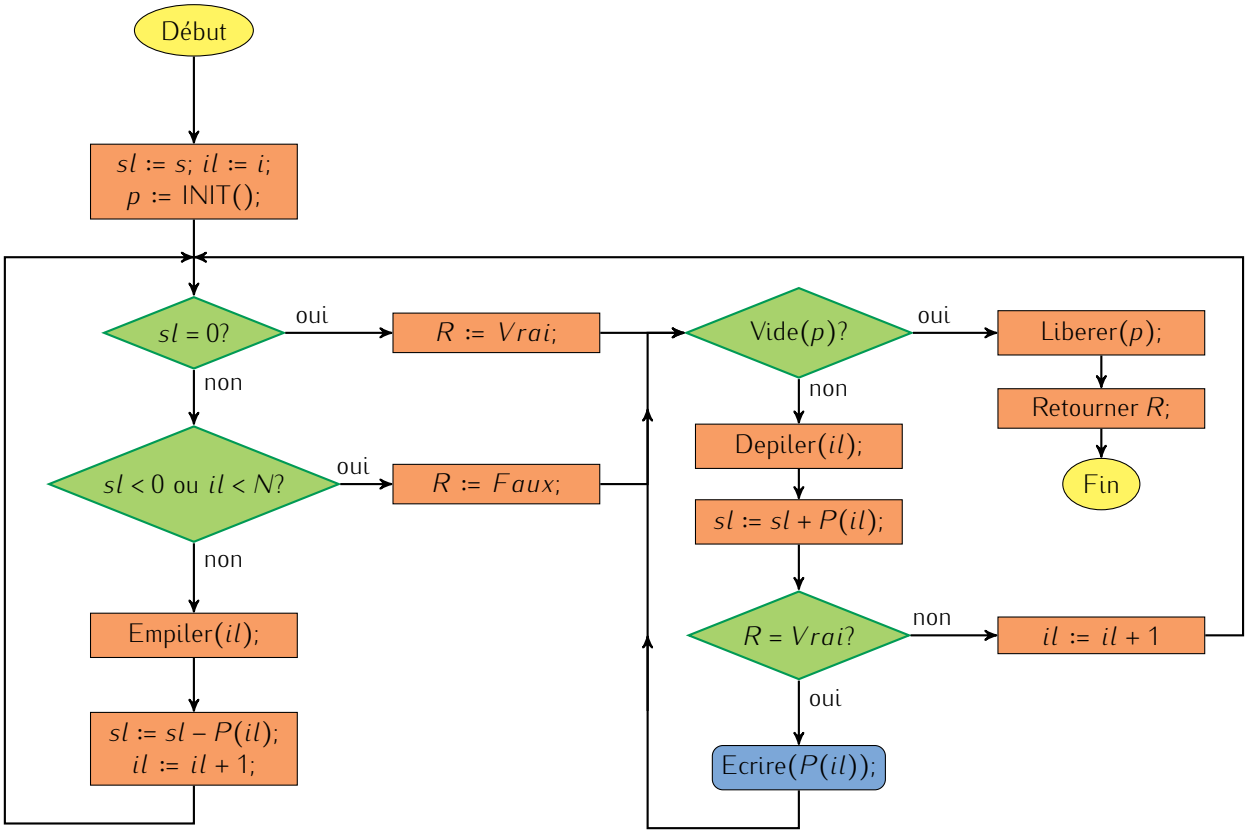
Logigramme initial



Suppression des appels terminaux



Suppression des appels non-terminaux



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 envoie sl dans la pile;
            On envoie 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

```
cCompilateur et options de compilation
n+nvCCo=gcc
n+nvCFLAGSo=-Wall -ansi -pedantic -Wextra -g

cFichiers du projet
n+nvSOURCESo=main.c stack.c truc.c
n+nvOBJECTSo=k(SOURCES:.co=.ok)

cNom du programme
n+nvEXECo=prog

k(EXECk): k(OBJECTSk)
	k(CCk) k(CFLAGSk) n+nv -o k(EXECk)

.c.o:
	k(CCk) -c k(CFLAGSk) n+nv*.c

clean:
	rm k(OBJECTSk) k(EXECk)
```

4.2 Jeux de tests

Exécution du programme avec le fichier suivant :