COMPTE RENDU DU TP2 DE SDD

Binôme :

AZEROUAL Mohammed

LINGOM NKAMGA David Cédric

Sommaire

1 Présentation générale 3
1.1 Description de l'objet du TP 3
1.2 Description de la structure de données 3
1.3 Schéma générale de la pile 3
1.4 Description des fichiers des données utilisées (en entrée et en sortie) 4
1.5 Organisation du code source 4
2 Détail de chaque fonction 5
2.1 Trace à la main de la fonction récursive 22
2.2 Processus de dérécursification 25
3 Compte rendu d'exécution 33
3.1 Makefile 33
3.2 Jeux de test complets 34
3.2.1 Jeux d'essais sur la fonction qui test les fonctions et procédures de pile 34
3.2.2 Jeux d'essais sur la fonction TrucRecursive 46
3.2.3 Jeux d'essais sur la fonction TrucIterative 52

1 Présentation générale

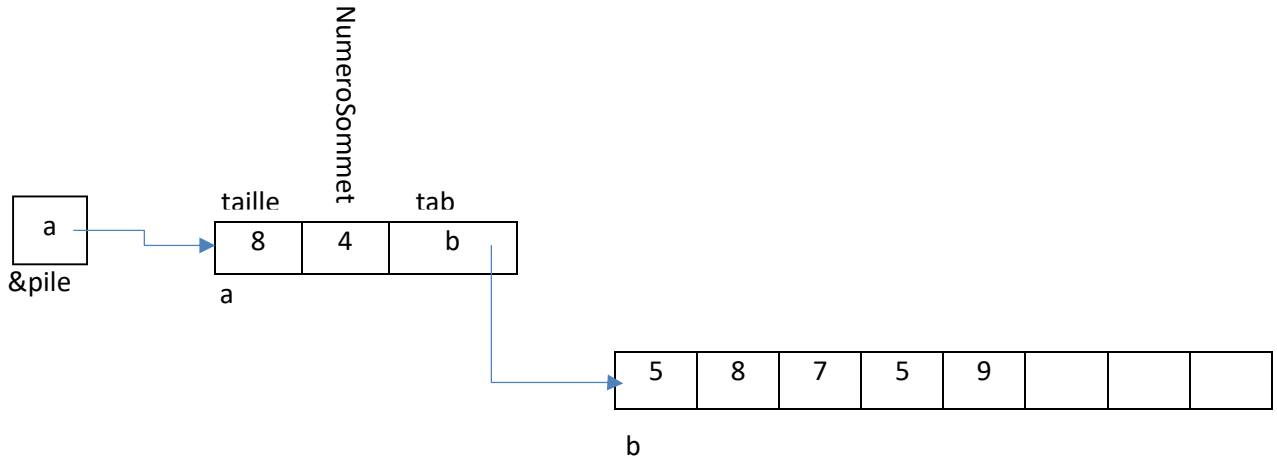
1.1 Description de l'objet du TP

Dans ce TP notre travail consiste à réaliser l'implémentation des fonctions et procédures de gestion de pile et les tester. Ensuite nous devrons implémenter une fonction récursive appelé « TRUC » ainsi que ça trace à la main et cette fonction récursive sera traduit sous forme itérative grâce aux fonctions et procédures de pile.

1.2 Description de la structure de données

Comme déjà dis plus haut la structure de données qui sera utilisé ici est une pile. Sa structure est telle que nous avons un pointeur sur une structure de pile dont le type est « pile_t » et dans cette structure nous avons trois champs . Le premier champ représente la taille de la pile, le deuxième champ « NumeroSommet » nous donne le numéro de l'élément au sommet de la pile, le troisième champ « tab » est un pointeur sur un tableau contenant les éléments de la pile qui sont de type « objet_t ». Le type « objet_t » est définit dans le fichier « pile.h ».

1.3 Schéma générale de la pile



1.4 Description des fichiers des données utilisées (en entrée et en sortie)

Les données qui seront dans le tableau seront traiter grâce la fonction « TRUC », soit sous forme récursive soit sous forme itérative seront. Ces seront données à partir de l'entrée standard grâce à la lecture au clavier et le résultat du traitement sera afficher en sortie standard.

1.5 Organisation du code source

Notre code source constitué de fichier.c et fichier.h qui sont comme suit:

Nous avons le fichier principal main.c qui contient notre fonction principale qui fait appelle aux procédures : « TestFonctionPile » (qui test les procédures et fonctions de pile) , « LectureTableau » (qui entrer les données dans le tableau) et sur ce tableau sera effectué un traitement grâce aux procédures « TrucRecursive » et « TrucIterative ». Notre fichier main.c contient comme fichier d'entête « truc.h » qui est aussi inclus dans dans le fichier « truc.c » et ce fichier d'entête contient les prototypes suivants :

```
void LectureTableau(objet_t *, int, int *);

void AfficherTableau(objet_t *, int);

void echanger(objet_t *, objet_t *);

void TrucRecursive(int, int, objet_t *);

void TrucIterative(int, int, objet_t *);
```

Ensuite dans le fichier d'entête « truc.h » est inclus le fichier d'entête « pile.h » qui elle est aussi inclus dans le fichier « pile.c » où sont définies les fonctions et les procédures liées à la gestion des piles dont les prototypes sont comme suit:

```
pile_t * InitialisationPile(int);

void EstPleine(pile_t *, enum BOOL *);

void empiler(pile_t *, int);

void EstVide(pile_t *, enum BOOL *);

objet_t depiler(pile_t *);

objet_t SommetPile(pile_t *);

void LibererPile(pile_t **);

void AfficherPile(pile_t *);

void TestFonctionPile(int *);
```

Outre cela dans le fichier d'entête « pile.h » sont inclus les fichiers d'entêtes standard « stdio.h » et « stdlib.h ».

2 Détail de chaque fonction

Au sujet des fonctions et procédures utilisés dans ce TP qui sont repartis dans les différents fichiers .c , nous allons vous donner les détails des fonctions et procédures de chaque fichier.c.

Les fonctions et procédures du fichier « pile.c » sont détaillés comme suit:

```
/*-------------------------------------------------------------------------------------------------*/
/* InitialisationPile        Alloue une structure pile ainsi que le tableau servant de pile.       */
/*                                                                                                 */
/* En entrée :      TaillePile - Elle représente la taille de la taille.                           */
/*                                                                                                 */
/* En sortie :      Renvoie l'adresse de la structure de pile.                                     */
/*                                                                                                 */
/* Variable(s) locale(s) :    pile - Pointeur sur la structure de pile.                            */
/*                                                                                                 */
/*-------------------------------------------------------------------------------------------------*/


pile_t * InitialisationPile (int TaillePile)
{
        pile_t * pile = (pile_t *)malloc(sizeof(pile_t));
        if(pile != NULL)                                    /* si l'allocation est reussite*/
        {
                pile->taille = TaillePile;                  /*fixer la taille de la pile*/
                pile->NumeroSommet = -1;
```

```
                pile->tab = (objet_t *)malloc(TaillePile * sizeof(objet_t));          /* allouer un tableau de donnees de taille TaillePile */
                if(pile->tab == NULL)                                   /* si l'allocation de tab est echouée*/
                 {
                   free(pile);                                          /* on libere la Pile */
                   pile = NULL;
                 }
        }
        return pile;
}



/*-------------------------------------------------------------------------------------------------------*/
/* EstPleine       Vérifie si notre pile est pleine.                                                     */
/*                                                                                                       */
/* En entrée :       pile - Pointeur sur la structure de pile.                                           */
/*                   pleine - Pointeur sur la case mémoire contenant vrai ou faux, pour dire */
/*                            comme quoi la pile est respectivement pleine ou pas.                       */
/*                                                                                                       */
/* En sortie :    pleine - Pointeur sur la case mémoire contenant vrai ou faux, pour                     */
/*                         dire comme quoi la pile est respectivement pleine ou pas.                     */
/*                                                                                                       */
/*-------------------------------------------------------------------------------------------------------*/
```

```
void EstPleine(pile_t * pile, enum BOOL * pleine)
{
        *pleine = vrai;                         /*suposons qu'elle est pleine*/
        if(pile->NumeroSommet < ((pile->taille)-1))         /*si la pile n'est pas pleine */
        {
                *pleine = faux;
        }
}



/*-------------------------------------------------------------------------------------------------  */
/* empiler       Empile une valeur dans la pile (Ajoute une valeur au sommet de la pile). */
/*                                                                                                    */
/* En entrée :    pile - Pointeur sur la structure de pile.                                           */
/*                 val - La valeur à empiler.                                                         */
/*                                                                                                    */
/* En sortie :    Rien en sortie.                                                                     */
/*                                                                                                    */
/*-------------------------------------------------------------------------------------------------*/



void empiler(pile_t * pile, objet_t val)
{
         (pile->tab)[pile->NumeroSommet + 1] = val;        /*ajouter la valeur à la pile */
         ++pile->NumeroSommet;                          /*augmenter le nombre d'element de la pile */
}
```

```
/*-------------------------------------------------------------------------------------------------*/
/* EstVide       Vérifie si la pile vide.                                                          */
/*                                                                                                 */
/* En entrée :      pile - Pointeur sur la structure de pile.                                      */
/*                     vide - Pointeur sur la case mémoire contenant comme valeur vrai ou          */
/*                               faux, pour dire comme quoi la pile est respectivement vide ou pas.*/
/*                                                                                                 */
/* En sortie :    vide - Pointeur sur la case mémoire contenant comme valeur vrai ou              */
/*                           faux, pour dire comme quoi la pile est respectivement vide ou pas.    */
/*                                                                                                 */
/*-------------------------------------------------------------------------------------------------*/


void EstVide(pile_t * pile , enum BOOL * vide)
{
        *vide = faux;
        if(pile->NumeroSommet == -1)
         {
           *vide = vrai ;
         }
}
/*-------------------------------------------------------------------------------------------------*/
/* depiler       Retire la valeur au sommet et la supprime de la pile.                             */
/*                                                                                                 */
/* En entrée :   pile - Pointeur de la structure de pile.                                          */
/*                                                                                                 */
/* En sortie :   Retourne le sommet de la pile.                                                    */
```

```
/*                                                                                                    */
/* Variable(s) locale(s) :    sommet - Contenant la valeur du sommet de la pile à être        */
/*                                          supprimer de la pile.                                    */
/*                                                                                                    */
/*----------------------------------------------------------------------------------------------------*/



objet_t depiler(pile_t * pile)
{
        objet_t sommet = (pile->tab)[pile->NumeroSommet];   /* sauvegarde le sommet*/
        --pile->NumeroSommet;                       /*diminue le nombre d'éléments de la pile */
        return sommet;
}
```

```
/*-------------------------------------------------------------------------------------------*/
/* SommetPile        Renvoie le sommet de la pile.                                           */
/*                                                                                           */
/* En entrée :    pile - Pointeur sur la structure de pile.                                  */
/*                                                                                           */
/* En sortie :    La fonction retourne le sommet de la pile.                                 */
/*                                                                                           */
/*-------------------------------------------------------------------------------------------*/


objet_t SommetPile(pile_t * pile)
{
        return (pile->tab)[pile->NumeroSommet];
}


/*-------------------------------------------------------------------------------------------*/
/* LibererPile       Libère la pile.                                                         */
/*                                                                                           */
/* En entrée :     ppile - Pointeur de pointeur sur la structure de pile.                    */
/*                                                                                           */
/* En sortie :     ppile - L'adresse du pointeur sur la structure de pile.                   */
/*                                                                                           */
/*-------------------------------------------------------------------------------------------*/
```

```
void LibererPile(pile_t ** ppile)
{
	free((*ppile)->tab);	/* désallouer le tab de donnees(implicitement la pile))*/
	free(*ppile);		/* désallouer la structure de pile */
	*ppile = NULL;
}


/*-------------------------------------------------------------------------------------------------*/
/* AfficherPile	Affiche le contenu de notre pile.				*/
/*								*/
/* En entrée :	pile - Pointeur sur la structure de pile.			*/
/*								*/
/* En sortie :	Rien en sortie.						*/
/*								*/
/* Variable(s) locale(s) : i - variable servant de variable de boucle.		*/
/*								*/
/*-------------------------------------------------------------------------------------------------*/

void AfficherPile(pile_t * pile)
{
	int i;
	for(i = pile->NumeroSommet; i > -1; --i)
	{
		printf("%d\n", (pile->tab)[i]);
	}
}
```

```
/*-------------------------------------------------------------------------------------------------*/
/* TestFonctionPile         Procédure permettant d'utiliser et test le bon fonctionnement  */
/*                          des fonctions et procédures de pile.                           */
/*                                                                                         */
/* En entrée :    CodeLecture - Pointeur sur la case contenant une valeur entière          */
/*                              comme quoi la lecture c'est bien passé ou pas.             */
/*                                                                                         */
/* En sortie :    CodeLecture - Pointeur sur la case contenant une valeur entière          */
/*                              comme quoi la lecture c'est bien passé ou pas.             */
/*                                                                                         */
/* Variable(s) locale(s) :       choix - Variable permettant de gérer le menu fait avec le */
/*                                       switch.                                           */
/*                             continuer - Variable permettant de faire choix de continuer */
/*                                       l'exécution ou pas.                               */
/*                               valeur - La valeur qui va être dans la pile.              */
/*                                 pile - Pointeur sur la structure de pile.               */
/*                                vide - Variable contenant vrai si la pile est vide et faux sinon. */
/*                               pleine - Variable contenant vrai si la pile est pleine et faux sinon*/
/*                                                                                         */
/*-------------------------------------------------------------------------------------------------*/
```

```
void TestFonctionPile(int * CodeLecture)
{
	int choix, continuer;
	objet_t valeur;
	pile_t * pile = InitialisationPile(TAILLE_MAX);
	enum BOOL vide, pleine;
	if(pile != NULL)				/*si l'allocation de la pile c'est bien passé*/
	{
		do
		{
			printf("Veuillez faire un choix!\n");
			printf("1. Tester si la pile est vide.\n");
			printf("2. Tester si la pile est pleine.\n");
			printf("3. Empiler.\n");
			printf("4. Depiler.\n");
			printf("5. Sommet de la pile.\n");
			*CodeLecture = scanf("%d", &choix);
			if(*CodeLecture)				/*si lecture c'est bien passé*/
			{
				switch(choix)
				{
				case 1:
					EstVide(pile,&vide);		/*cherche si la pile est vide*/
					if(vide)			/*si elle vide*/
					{
						printf(" La pile est vide. ");
					}
```

```
        else                                    /*si elle n'est pas vide*/
        {
                printf("La pile n'est pas vide.\n");
        }
        break;
case 2:
        EstPleine(pile,&pleine);            /*vérifie si la pile est pleine*/
        if(pleine)                              /*si elle est pleine*/
        {
                printf(" La pile est pleine. ");
        }
        else                                    /*si elle n'est pas pleine*/
        {
                printf("La pile n'est pas pleine.\n");
        }
        break;
case 3:
        EstPleine(pile, &pleine);           /*vérifie si la pile est pleine*/
        if(!pleine)                             /*si elle n'est pas pleine*/
        {
                printf("Le cotenu de la pile avant d'empiler.\n");
                AfficherPile(pile);
                printf("Veuillez entrer le chiffre à empiler\n");
                *CodeLecture = scanf("%d", &valeur);
                if(*CodeLecture)        /*si la lecture c'est bien passé*/
                {
                        empiler(pile, valeur);      /*empiler la valeur*/
```

```
                printf("Le cotenu de la pile après avoir empiler.\n");
                AfficherPile(pile);
            }
        }
        else
        {
            printf("La pile est pleine.\n");
        }
        break;
    case 4:
        EstVide(pile, &vide);          /*vérifie si la pile est vide*/
        if(!vide)                      /*si elle n'est pas vide*/
        {
            printf("Le cotenu de la pile avant d'empiler.\n\n");
            AfficherPile(pile);
            printf("La valeur dépiler est:\n %d\n\n", depiler(pile));
            printf("Le cotenu de la pile après avoir depiler.\n\n");
            AfficherPile(pile);
        }
        else
        {
            printf("La pile est vide.\n");
        }
        break;
    case 5:
        EstVide(pile, &vide);          /*vérifie si la pile est vide*/
```

```
                                        if(!vide)                               /*si la pile n'est pas vide*/
                                        {
                                                printf("Le cotenu de la pile avant d'empiler.\n\n");
                                                AfficherPile(pile);
                                                printf("Le sommet de la pile est:\n %d\n",
SommetPile(pile));
                                        }
                                        else
                                        {
                                                printf("La pile est vide.\n");
                                        }
                                        break;
                                        default:
                                        printf("Veuillez entrer le bon numéro!\n");
                                }
                        }
                        printf("\nVoulez-vous continuer? 1 = OUI  0 = NON\n");
                        *CodeLecture = scanf("%d", &continuer);
                }while((continuer && *CodeLecture) && (pile->NumeroSommet < TAILLE_MAX));
                LibererPile(&pile);
        }
}
```

Les fonctions et procédures du fichier « truc.c » sont détaillés comme suit:

```
/*-------------------------------------------------------------------------------------------------*/
/*                                          truc.c                                                 */
/*                                                                                                 */
/* Role : Définition des fonctions et procédures permettant de générer toutes                      */
 /*        les permutations d'un ensemble de valeurs de manière récursive et de                    */
/*        manière itérative.                                                                       */
/*                                                                                                 */
/*-------------------------------------------------------------------------------------------------*/


#include "./truc.h"


/*-------------------------------------------------------------------------------------------------*/
/* LectureTableau   Permet d'entrer les vaeurs qui vont etre permuter dans le                      */
/*                  tableau.                                                                       */
/*                                                                                                 */
/* En entrée :          tab - Pointeur sur un tableau de valeurs.                                  */
/*                     taille - La taille du tableau.                                              */
/*              CodeLecture - Pointeur sur la case contenant une valeur                            */
/*                            permettant de déterminer si lecture à réussit ou                     */
/*                            pas.                                                                 */
/*                                                                                                 */
/* En sortie :  CodeLecture - Pointeur sur la case contenant une valeur                            */
/*                            permettant de déterminer si lecture à reussit ou                     */
/*                            pas.                                                                 */
```

```
/*                                                                                                  */
/* Variable(s) locale(s) :     i - Variable servant de compteur dans la boucle.                     */
/*                                                                                                  */
/*--------------------------------------------------------------------------------------------------*/


void LectureTableau(objet_t * tab, int taille, int * CodeLecture)
{
        int i;
        printf("Veuillez les éléments à permuter dans le tableau.\n");
        for(i=0; (i < taille) && (*CodeLecture); ++i)
        {
                printf("Veuillez entrer l'élément %d\n", i + 1);
                *CodeLecture = scanf("%d", tab + i);
        }
}


/*--------------------------------------------------------------------------------------------------*/
/* AfficherTableau    Affiche les valeurs contenu dans le tableau.                                  */
/*                                                                                                  */
/* En entrée :        tab - Pointeur sur un tableau de valeurs.                                     */
/*                    taille - La taille du tableau.                                                */
/*                                                                                                  */
/* En sortie :     Rien en sortie.                                                                  */
/*                                                                                                  */
/* Variable(s) locale(s) :     j - Variable servant de compteur dans la boucle.                     */
```

```
/*                                                                                                    */
/*----------------------------------------------------------------------------------------------------*/



void AfficherTableau(objet_t * tab, int taille)
{
        int j;
        for(j=0; j < taille; ++j)
        {
                printf("%d\t", tab[j]);
        }
        printf("\n");
}



/*----------------------------------------------------------------------------------------------------*/
/* echanger         Fait l'échange entre deux valeurs.                                                */
/*                                                                                                    */
/* En entrée :     a - Pointeur sur la première valeur.                                               */
/*                  b - Pointeur sur la deuxième valeur.                                              */
/*                                                                                                    */
/* En sortie :     a - Pointeur sur la première valeur.                                               */
/*                 b - Pointeur sur la deuxième valeur.                                               */
/*                                                                                                    */
/* Variable(s) locale(s) :     temp - Servant de variable temporaire pour l'échange.                  */
/*                                                                                                    */
/*----------------------------------------------------------------------------------------------------*/
```

```
void echanger(objet_t * a, objet_t * b)
{
	objet_t temp;
	temp = *a;
	*a = *b;
	*b = temp;
}


/*-------------------------------------------------------------------------------------------------*/
/* TrucRecursive	Donne toutes les permutation des valeurs contenu dans le	*/
/*			tableau.					*/
/*									*/
/* En entrée :	i - Servant dans la condition d'affichage et d'initialisation de	*/
/*			boucle.						*/
/*		n - La taille du tableau.				*/
/*		tab - Pointeur sur le tableau de valeurs qui devront être permuté.	*/
/*									*/
/* En sortie :	Rien en sortie.						*/
/*									*/
/* Variable(s) locale(s) :	j - Variable servant de compteur de boucle.	*/
/*									*/
/*-------------------------------------------------------------------------------------------------*/
```

```
void TrucRecursive(int i, int n, objet_t * tab)
{
	int j;
	if(i == n)
	{
		AfficherTableau(tab, n);
	}
	else
	{
		for(j = i; j <= n; ++j)
		{
			echanger(tab + (i-1), tab + (j-1));
			TrucRecursive(i + 1, n, tab);
			echanger(tab + (i-1), tab + (j-1));
		}
	}
}
```

2.1 Trace à la main de la fonction récursive

La Trace TrucRecursive(1, n, T)

On pose T= [4,5,6]

TrucRcursive(1,3, T) :

J=1 :

T= [4, 5, 6] (ce tableau est obtenu après la permutation de la case j=1 avec i=1)

TrucRecursive(2,3, T) :

J=2 :

T= [4,5,6] (ce tableau est obtenu après la permutation de la case j=2 avec i=2)

TrucRecursive(3, 3,T) :

Ecrire : 4 5 6

T= [4, 5, 6] (ce tableau est obtenu après la permutation de la case j=2 avec i=2)

J=3 :

T= [4, 6, 5] (ce tableau est obtenu après la permutation de la case j=3 avec i=2)

TrucRecursive(3,3 ,T) :

Ecrire : 4 6 5

T= [4, 5, 6] (ce tableau est obtenu après la permutation de la case j=3 avec i=2)

T= [4, 5, 6] (ce tableau est obtenu après la permutation de la case j=1 avec i=1)

J=2 :

T= [5, 4, 6] (ce tableau est obtenu après la permutation de la case j=2 avec i=1)

TrucRecursive(2,3, T) :

J=2 :

T= [5, 4, 6] (ce tableau est obtenu après la permutation de la case j=2 avec i=2)

TrucRecursive(3,3 ,T) :

Ecrire : 5 4 6

T= [5, 4, 6] (ce tableau est obtenu après la permutation de la case j=2 avec i=2)

J=3 :

T= [5, 6, 4] (ce tableau est obtenu après la permutation de la case j=3 avec i=2)

TrucRecursive(3,3,T) :

Ecrire : 5 6 4

T= [5, 4, 6] (ce tableau est obtenu après la permutation de la case j=3 avec i=2)

T= [4, 5, 6] (ce tableau est obtenu après la permutation de la case j=2 avec i=1)

J=3 :

T= [6, 5, 4] (ce tableau est obtenu après la permutation de la case j=3 avec i=1)

TrucRecursive(2,3, T) :

J=2 :

T= [6, 5, 4] (ce tableau est obtenu après la permutation de la case j=2 avec i=2)

TrucRecursive(3,3, T) :

Ecrire : 6 5 4

T= [6, 5,4] (ce tableau est obtenu après la permutation de la case j=2 avec i=2)

J=3 :

T= [6, 4, 5] (ce tableau est obtenu après la permutation de la case j=3 avec i=2)

TrucRecursive(3,3,T) :

Ecrire : 6 4 5

T= [6, 5 , 4] (ce tableau est obtenu après la permutation de la case j=3 avec i=2)

T= [4, 5 , 6] (ce tableau est obtenu après la permutation de la case j=3 avec i=1)

D'après la trace effectué à la main on constate que la fonction récursive « TrucRecursive » permet de donner toutes les permutations des valeurs dans le tableau. Le nombre de permutation est donné par la formule$\big((n-i)+1\big)!$. Donc pour cette exécution vue que nous avons pris $i=1\ et\ n=3$ nous le nombre de permutations est $\big((3-1)+1\big)!=3!=6\ permutations$.

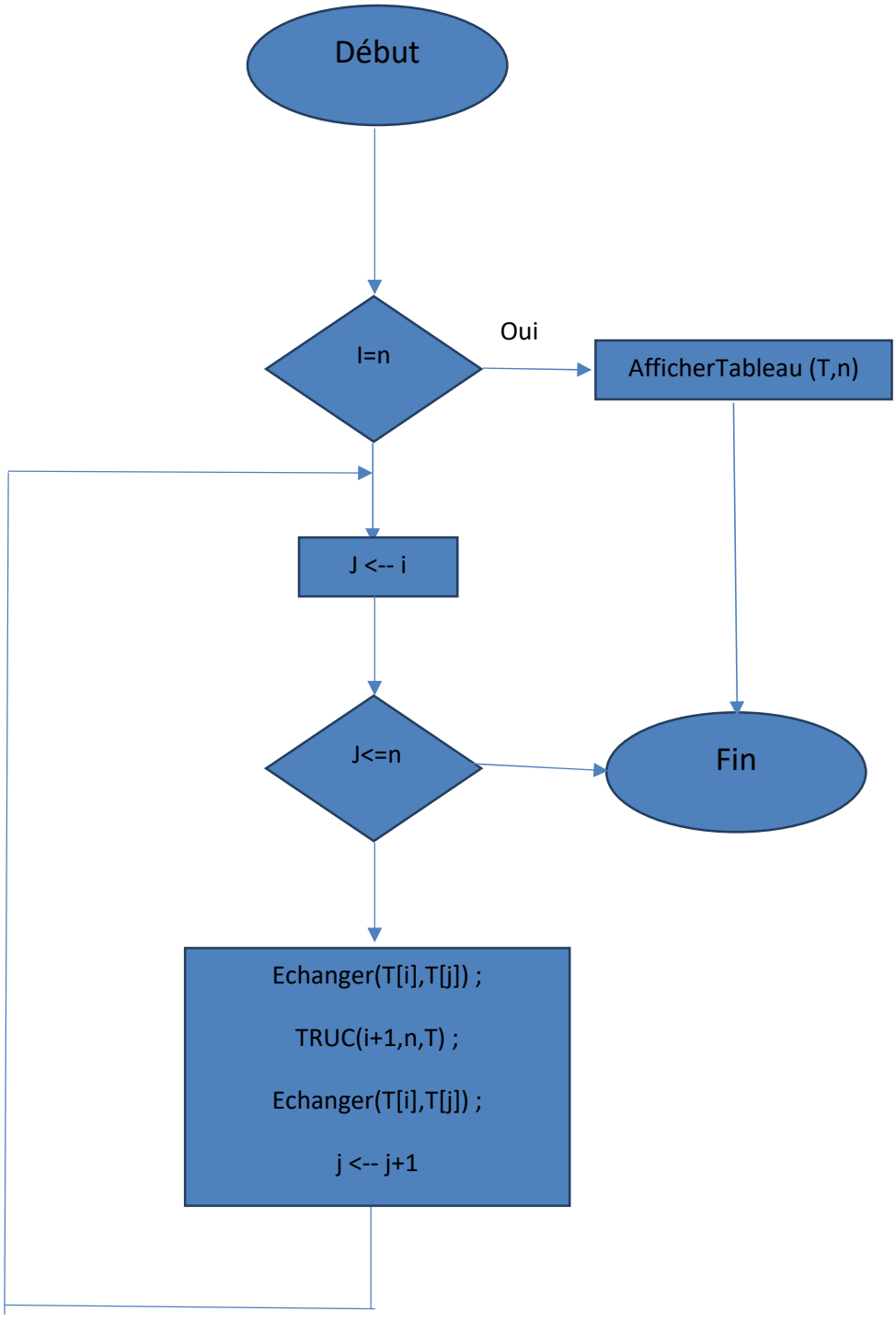
2.2 Processus de dérécursification

Ce processus ce déroule principalement en trois c'est-à-dire :

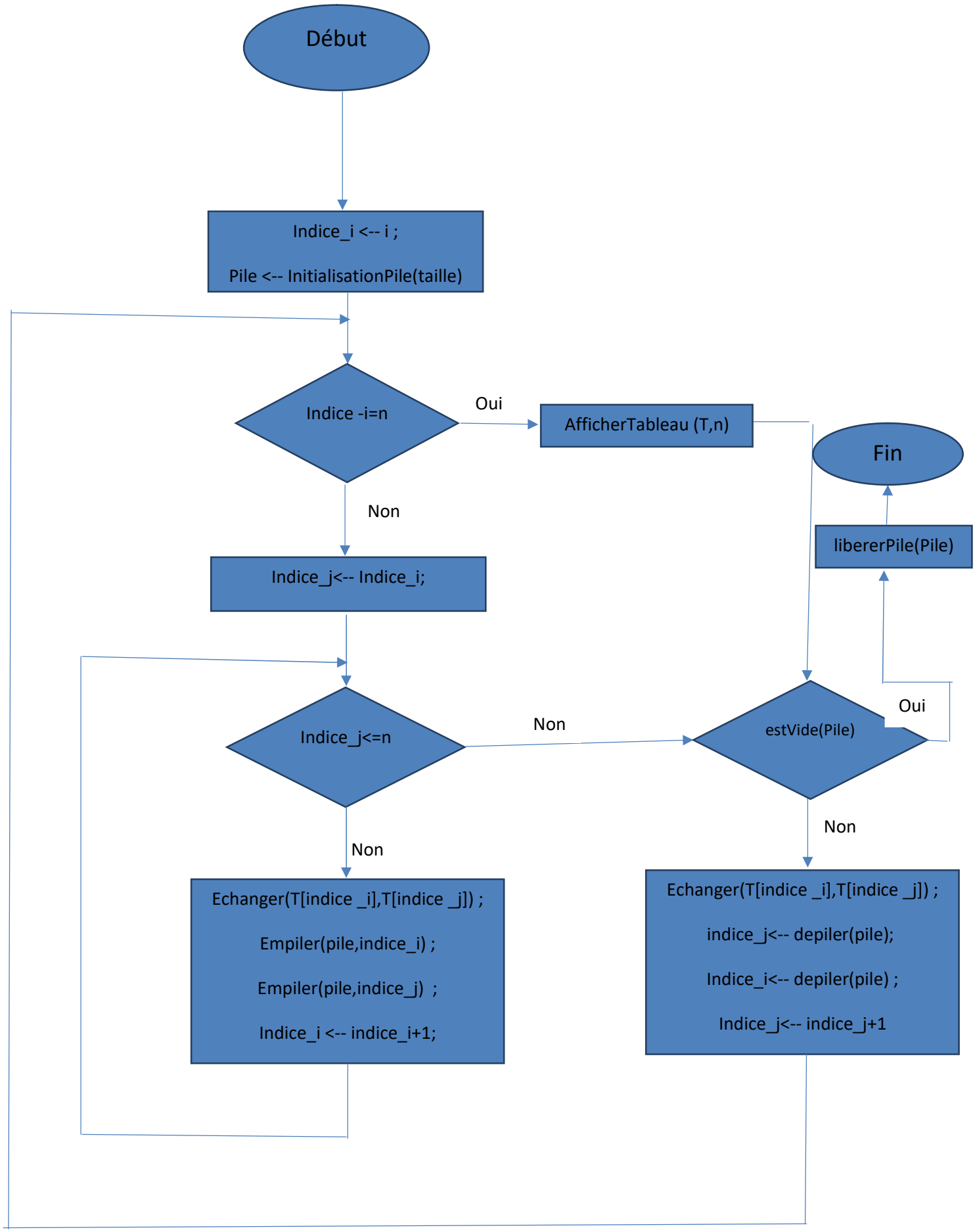
- Etape 0 : Réécriture de la fonction récursive sous forme de digramme
- Etape 1 : Suppression de l'appel terminal
- Etape 2 : suppression de l'appel non terminal

Mais que vue dans notre fonction récursive nous avons qu'un appel non terminal nous nous concentrerons sur l'étape 0 et l'étape 2.

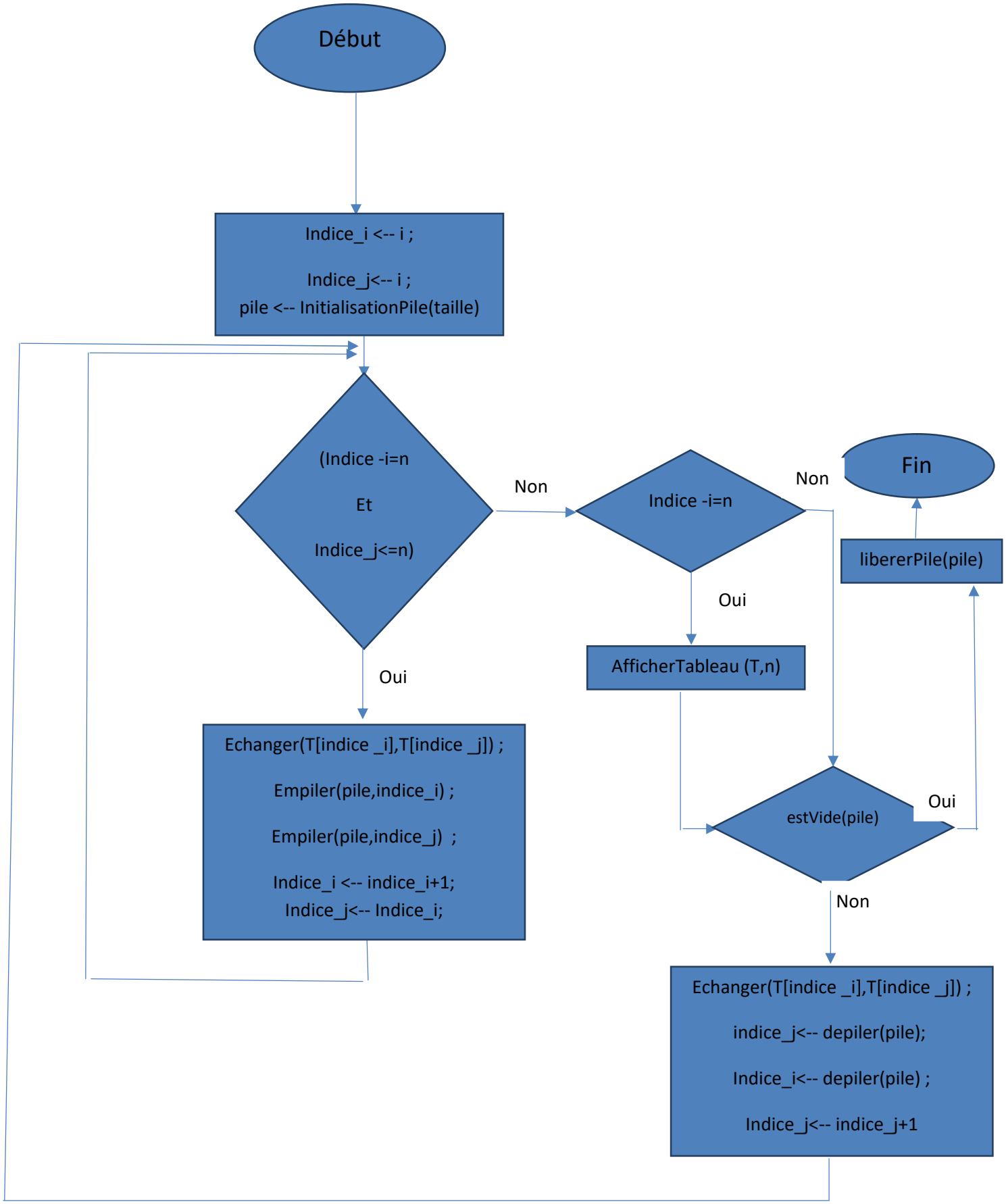
Etape 1 : Réécriture de la fonction récursive sous forme de digramme



Etape 2 : Suppression de l'appel non terminal



Etape 2(améliorer) : Suppression de l'appel non terminal



```
/*-------------------------------------------------------------------------------------------------*/
/* TrucIterative       Donne toutes les permutation des valeurs contenu dans le tableau.  */
/*                                                                                                 */
/* En entrée :         i - Indice de départ dans le tableau.                                       */
/*                     n - La taille  du tableau.                                                  */
/*                   tab - Pointeur sur le tableau des valeurs.                                    */
/*                                                                                                 */
/* En sortie :    Rien en sortie.                                                                  */
/*                                                                                                 */
/* Variable(s) locale(s) :    indice_i - Variable servant de condition de boucle.                  */
/*                            indice_j - Variable servant de condition de boucle.                  */
/*                                 fin - Variable contenant vrai pour dire la pile vide donc       */
/*                                       nous avons donné toutes les permutations.                 */
/*                                       Par contre faux veux la pile n'est pas encore vide        */
/*                                       nous n’avons pas donné toutes les permutations.           */
/*                               pile - Pointeur sur la structure de pile.                         */
/*                                                                                                 */
/*-------------------------------------------------------------------------------------------------*/
```

```
void TrucIterative(int i, int n, objet_t * tab)
{
	objet_t indice_i, indice_j;
	enum BOOL fin;
	pile_t * pile;
	indice_i = i;
	indice_j = i;
	fin = faux;
	pile = InitialisationPile(2*(n-1));
	if(pile != NULL)			/*pile alloué et bien initialiser*/
	{
		while(!fin)
		{
			while((indice_i < n) && (indice_j <= n))
			{
				echanger(tab + (indice_i - 1), tab + (indice_j - 1));
				empiler(pile, indice_i);
				empiler(pile, indice_j);
				++indice_i;
				indice_j = indice_i;
			}
			if(indice_i == n)
			{
				AfficherTableau(tab, n);
			}
			EstVide(pile, &fin);			/*vérifie si la pile est vide*/
			if(!fin)
```

```
                {
                        indice_j = depiler(pile);
                        indice_i = depiler(pile);
                        echanger(tab + (indice_i - 1), tab + (indice_j -1));
                        ++indice_j;
                }
        }
        LibererPile(&pile);
    }
}
```

Le fichier main.c qui contient notre fonction principal faisant appel autres fonctions citez plu haut est comme suit:

```
/*-------------------------------------------------------------------------------------------------*/
/*                                          main.c                                                 */
/*                                                                                                 */
/* Role : La fonction principale faisant appelle à des fonctions qui test les fonctions,           */
/*        qui donne qui donne toute les permutations de manière récursive et aussi                 */
/*        toutes les permutations de manière itérative. Ces fonctions ne sont pas                  */
/*        appelé toutes en meme temps mais plutot au choix grace au ménu.                          */
/*                                                                                                 */
/*-------------------------------------------------------------------------------------------------*/



#include "./truc.h"
```

```
int main()
{
	int choix, continuer, CodeLecture = 1;
	objet_t tab[TAILLETAB];
	do
	{
		printf("Que voulez vous faire?\n\n");
		printf("1. Tester les fonctions et procédures de pile.\n");
		printf("2. Exécuter la version récursive de TRUC.\n");
		printf("3. Exécuter la version itérative de TRUC.\n");
		CodeLecture = scanf("%d", &choix);
		if(CodeLecture)				/*si la lecture c'est bien passé*/
		{
			switch(choix)
			{
			case 1:
				TestFonctionPile(&CodeLecture);
				if(!CodeLecture)
				{
					printf("Erreur de lecture!\n");
				 }
				break;
			case 2:
				LectureTableau(tab, TAILLETAB, &CodeLecture);
				if(CodeLecture)
				{
					TrucRecursive(1, TAILLETAB, tab);
```

```
                }
                else
                {
                        printf("Erreur de lecture.\n");
                }
                break;
        case 3:
                LectureTableau(tab, TAILLETAB, &CodeLecture);
                if(CodeLecture)
                {
                        TrucIterative(1, TAILLETAB, tab);
                }
                else
                {
                        printf("Erreur de lecture.\n");
                }
                break;
        default:
                printf("Vous avez fait le mauvais choix.\n");
        }
    }
    printf("Voulez vous continuer? 1 = OUI   0 = NON\n\n");
    CodeLecture = scanf("%d", &continuer);
}while(continuer && CodeLecture);
return 0;
}
```

3 Compte rendu d'exécution

3.1 Makefile

```
#compilateur

CC = gcc

#Profileur

VG = valgrind

#Option

CFLAGS = -ansi -pedantic -Wall -Wextra -g -O2
LDFLAGS = -lm

#liste des fichiers objets

OBJ = main.o pile.o truc.o


#Règle d'éxécution
execute : main
    ./main

#Regle de production de l'exécutable

main : $(OBJ)
    $(CC) -o main $(OBJ) $(CFLAGS) $(LDFLAGS)


#Règle de production des fichiers objets


main.o: main.c truc.h
    $(CC) -c main.c $(CFLAGS) $(LDFLAGS)
```

```
pile.o: pile.c pile.h
	$(CC) -c pile.c $(CFLAGS) $(LDFLAGS)

truc.o: truc.c truc.h
	$(CC) -c truc.c $(CFLAGS) $(LDFLAGS)

#Regle pour nettoyer

clean :
	rm $(OBJ) *.*~

#Regle d'utilisation de valgrind

valgrind:
	$(VG) ./main
```

3.2 Jeux de test complets

Pour ses jeux d'essais, tous les données entrée et sortie figure dans le shell c'est-à-dire en entrée standard et sortie standard.

3.2.1 Jeux d'essais sur la fonction qui test les fonctions et procédures de pile

3.2.1.1 Empiler dans une pile pas pleine

- **Résultat de l'exécution**

```
Veuillez faire un choix!
1. Tester si la pile est vide.
2. Tester si la pile est pleine.
3. Empiler.
4. Depiler.
5. Sommet de la pile.
3
Le cotenu de la pile avant d'empiler.
3
2
1
Veuillez entrer le chiffre à empiler
4
Le cotenu de la pile après avoir empiler.
4
3
2
1
```

- **Exécution avec valgrind**

```
Veuillez faire un choix!
1. Tester si la pile est vide.
2. Tester si la pile est pleine.
3. Empiler.
4. Depiler.
5. Sommet de la pile.
3
Le cotenu de la pile avant d'empiler.
3
2
1
Veuillez entrer le chiffre à empiler
4
Le cotenu de la pile après avoir empiler.
4
3
2
1

Voulez-vous continuer? 1 = OUI  0 = NON
0
Voulez vous continuer? 1 = OUI   0 = NON

0
==2529==
==2529== HEAP SUMMARY:
==2529==     in use at exit: 0 bytes in 0 blocks
==2529==   total heap usage: 2 allocs, 2 frees, 56 bytes allocated
==2529==
==2529== All heap blocks were freed -- no leaks are possible
==2529==
==2529== For counts of detected and suppressed errors, rerun with: -v
==2529== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3.2.1.2 Empiler dans une pile pleine

- **Résultat de l'exécution**

- **L'exécution avec valgrind**

```
Le cotenu de la pile après avoir empiler.
10
9
8
7
6
5
4
3
2
1

Voulez-vous continuer? 1 = OUI  0 = NON
1
Veuillez faire un choix!
1. Tester si la pile est vide.
2. Tester si la pile est pleine.
3. Empiler.
4. Depiler.
5. Sommet de la pile.
3
La pile est pleine.

Voulez-vous continuer? 1 = OUI  0 = NON
0
Voulez vous continuer? 1 = OUI   0 = NON

0
==2572==
==2572== HEAP SUMMARY:
==2572==     in use at exit: 0 bytes in 0 blocks
==2572==   total heap usage: 2 allocs, 2 frees, 56 bytes allocated
==2572==
==2572== All heap blocks were freed -- no leaks are possible
==2572==
==2572== For counts of detected and suppressed errors, rerun with: -v
==2572== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.1.3 Dépiler dans un pile vide

- **Résultat de l'exécution**

- **L'exécution avec valgrind**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
1
Veuillez faire un choix!
1. Tester si la pile est vide.
2. Tester si la pile est pleine.
3. Empiler.
4. Depiler.
5. Sommet de la pile.
4
La pile est vide.

Voulez-vous continuer? 1 = OUI  0 = NON
0
Voulez vous continuer? 1 = OUI   0 = NON

0
==2635==
==2635== HEAP SUMMARY:
==2635==     in use at exit: 0 bytes in 0 blocks
==2635==   total heap usage: 2 allocs, 2 frees, 56 bytes allocated
==2635==
==2635== All heap blocks were freed -- no leaks are possible
==2635==
==2635== For counts of detected and suppressed errors, rerun with: -v
==2635== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.1.4 Dépiler dans une pile non vide

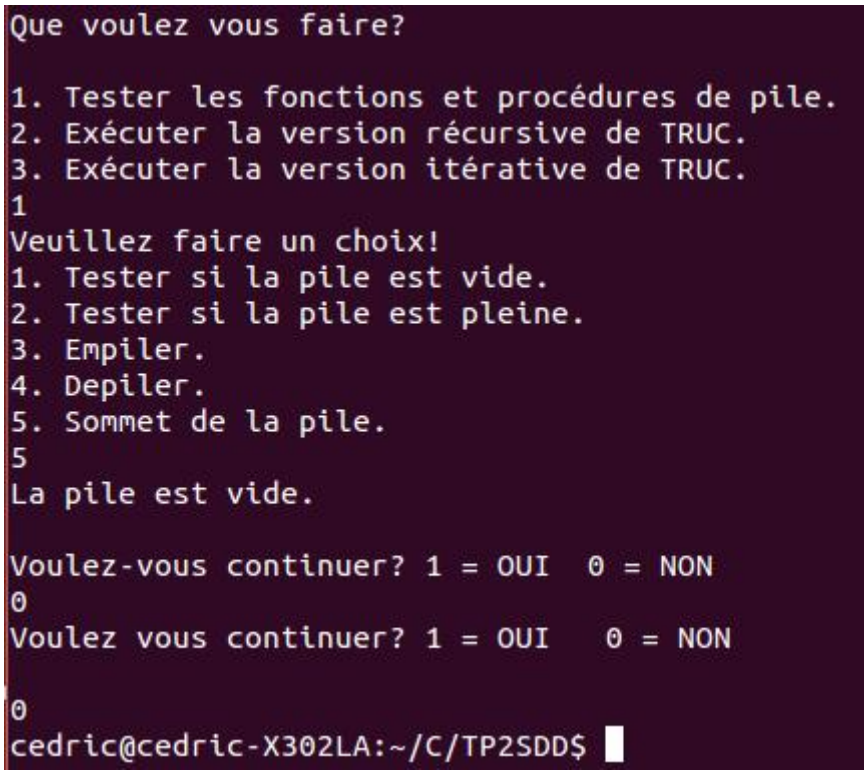
- **Résultat de l'exécution**

- **L'exécution avec valgrind**

```
Veuillez faire un choix!
1. Tester si la pile est vide.
2. Tester si la pile est pleine.
3. Empiler.
4. Depiler.
5. Sommet de la pile.
4
Le cotenu de la pile avant de dépiler.

4
3
2
1
La valeur dépiler est:
 4

Le cotenu de la pile après avoir dépiler.

3
2
1

Voulez-vous continuer? 1 = OUI  0 = NON
0
Voulez vous continuer? 1 = OUI   0 = NON

0
==2695==
==2695== HEAP SUMMARY:
==2695==     in use at exit: 0 bytes in 0 blocks
==2695==   total heap usage: 2 allocs, 2 frees, 56 bytes allocated
==2695==
==2695== All heap blocks were freed -- no leaks are possible
==2695==
==2695== For counts of detected and suppressed errors, rerun with: -v
==2695== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.1.5 Retourner le sommet de la pile lorsqu'elle est vide

- **Résultat de l'exécution**



- **Exécution avec valgrind**

```
cedric@cedric-X302LA:~/C/TP2SDD$ make valgrind
valgrind ./main
==2748== Memcheck, a memory error detector
==2748== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2748== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2748== Command: ./main
==2748==
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
1
Veuillez faire un choix!
1. Tester si la pile est vide.
2. Tester si la pile est pleine.
3. Empiler.
4. Depiler.
5. Sommet de la pile.
5
La pile est vide.

Voulez-vous continuer? 1 = OUI  0 = NON
0
Voulez vous continuer? 1 = OUI   0 = NON

0
==2748==
==2748== HEAP SUMMARY:
==2748==     in use at exit: 0 bytes in 0 blocks
==2748==   total heap usage: 2 allocs, 2 frees, 56 bytes allocated
==2748==
==2748== All heap blocks were freed -- no leaks are possible
==2748==
==2748== For counts of detected and suppressed errors, rerun with: -v
==2748== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.1.6 Retourner le sommet de la pile lorsqu'elle non vide

- **Résultat de l'exécution**

```
Veuillez faire un choix!
1. Tester si la pile est vide.
2. Tester si la pile est pleine.
3. Empiler.
4. Depiler.
5. Sommet de la pile.
5
Le cotenu de la pile est:

3
2
1
Le sommet de la pile est:
 3

Voulez-vous continuer? 1 = OUI   0 = NON
0
Voulez vous continuer? 1 = OUI    0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
Veuillez faire un choix!
1. Tester si la pile est vide.
2. Tester si la pile est pleine.
3. Empiler.
4. Depiler.
5. Sommet de la pile.
5
Le cotenu de la pile est:

3
2
1
Le sommet de la pile est:
 3

Voulez-vous continuer? 1 = OUI  0 = NON
0
Voulez vous continuer? 1 = OUI   0 = NON

0
==2815==
==2815== HEAP SUMMARY:
==2815==     in use at exit: 0 bytes in 0 blocks
==2815==   total heap usage: 2 allocs, 2 frees, 56 bytes allocated
==2815==
==2815== All heap blocks were freed -- no leaks are possible
==2815==
==2815== For counts of detected and suppressed errors, rerun with: -v
==2815== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.2 Jeux d'essais sur la fonction TrucRecursive

3.2.2.1 Lorsque i=1 et n=4

- **Résultat de l'exécution**

```
1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
2
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
1       2       4       3
1       3       2       4
1       3       4       2
1       4       3       2
1       4       2       3
2       1       3       4
2       1       4       3
2       3       1       4
2       3       4       1
2       4       3       1
2       4       1       3
3       2       1       4
3       2       4       1
3       1       2       4
3       1       4       2
3       4       1       2
3       4       2       1
4       2       3       1
4       2       1       3
4       3       2       1
4       3       1       2
4       1       3       2
4       1       2       3
Voulez vous continuer? 1 = OUI    0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
1       2       3       4
1       2       4       3
1       3       2       4
1       3       4       2
1       4       3       2
1       4       2       3
2       1       3       4
2       1       4       3
2       3       1       4
2       3       4       1
2       4       3       1
2       4       1       3
3       2       1       4
3       2       4       1
3       1       2       4
3       1       4       2
3       4       1       2
3       4       2       1
4       2       3       1
4       2       1       3
4       3       2       1
4       3       1       2
4       1       3       2
4       1       2       3
Voulez vous continuer? 1 = OUI   0 = NON

0
==2902==
==2902== HEAP SUMMARY:
==2902==     in use at exit: 0 bytes in 0 blocks
==2902==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==2902==
==2902== All heap blocks were freed -- no leaks are possible
==2902==
==2902== For counts of detected and suppressed errors, rerun with: -v
==2902== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.2.2 Lorsque 1 < i< n et n=4 (ici i = 2)

- **Résultat de l'exécution**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
2
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
1       2       4       3
1       3       2       4
1       3       4       2
1       4       3       2
1       4       2       3
Voulez vous continuer? 1 = OUI   0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
2
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
1       2       4       3
1       3       2       4
1       3       4       2
1       4       3       2
1       4       2       3
Voulez vous continuer? 1 = OUI   0 = NON

0
==3031==
==3031== HEAP SUMMARY:
==3031==     in use at exit: 0 bytes in 0 blocks
==3031==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==3031==
==3031== All heap blocks were freed -- no leaks are possible
==3031==
==3031== For counts of detected and suppressed errors, rerun with: -v
==3031== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.2.3 Lorsque i=n et n=4

- **Résultat de l'exécution**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
2
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
Voulez vous continuer? 1 = OUI    0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
2
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
Voulez vous continuer? 1 = OUI    0 = NON

0
==3189==
==3189== HEAP SUMMARY:
==3189==     in use at exit: 0 bytes in 0 blocks
==3189==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==3189==
==3189== All heap blocks were freed -- no leaks are possible
==3189==
==3189== For counts of detected and suppressed errors, rerun with: -v
==3189== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.2.4 Lorsque i > n et n = 4

- **Résultat de l'exécution**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
2
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
Voulez vous continuer? 1 = OUI    0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
2
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
Voulez vous continuer? 1 = OUI    0 = NON

0
==3280==
==3280== HEAP SUMMARY:
==3280==     in use at exit: 0 bytes in 0 blocks
==3280==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==3280==
==3280== All heap blocks were freed -- no leaks are possible
==3280==
==3280== For counts of detected and suppressed errors, rerun with: -v
==3280== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.3 Jeux d'essais sur la fonction TrucIterative

3.2.3.1 Loresque i = 1 et n = 4

- **Résultat de l'exécution**

```
1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
3
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
1       2       4       3
1       3       2       4
1       3       4       2
1       4       3       2
1       4       2       3
2       1       3       4
2       1       4       3
2       3       1       4
2       3       4       1
2       4       3       1
2       4       1       3
3       2       1       4
3       2       4       1
3       1       2       4
3       1       4       2
3       4       1       2
3       4       2       1
4       2       3       1
4       2       1       3
4       3       2       1
4       3       1       2
4       1       3       2
4       1       2       3
Voulez vous continuer? 1 = OUI   0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
1       2       3       4
1       2       4       3
1       3       2       4
1       3       4       2
1       4       3       2
1       4       2       3
2       1       3       4
2       1       4       3
2       3       1       4
2       3       4       1
2       4       3       1
2       4       1       3
3       2       1       4
3       2       4       1
3       1       2       4
3       1       4       2
3       4       1       2
3       4       2       1
4       2       3       1
4       2       1       3
4       3       2       1
4       3       1       2
4       1       3       2
4       1       2       3
Voulez vous continuer? 1 = OUI   0 = NON

0
==2980==
==2980== HEAP SUMMARY:
==2980==     in use at exit: 0 bytes in 0 blocks
==2980==   total heap usage: 2 allocs, 2 frees, 40 bytes allocated
==2980==
==2980== All heap blocks were freed -- no leaks are possible
==2980==
==2980== For counts of detected and suppressed errors, rerun with: -v
==2980== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.3.2 Lorsque 1 < i <n et n=4 (ici i = 2)

- **Résultat de l'exécution**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
3
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
1       2       4       3
1       3       2       4
1       3       4       2
1       4       3       2
1       4       2       3
Voulez vous continuer? 1 = OUI    0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
3
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
1       2       4       3
1       3       2       4
1       3       4       2
1       4       3       2
1       4       2       3
Voulez vous continuer? 1 = OUI   0 = NON

0
==3080==
==3080== HEAP SUMMARY:
==3080==     in use at exit: 0 bytes in 0 blocks
==3080==   total heap usage: 2 allocs, 2 frees, 40 bytes allocated
==3080==
==3080== All heap blocks were freed -- no leaks are possible
==3080==
==3080== For counts of detected and suppressed errors, rerun with: -v
==3080== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.3.3 Lorsque i = n et n = 4

- **Résultat de l'exécution**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
3
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
Voulez vous continuer? 1 = OUI    0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
3
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
1       2       3       4
Voulez vous continuer? 1 = OUI    0 = NON

0
==3230==
==3230== HEAP SUMMARY:
==3230==     in use at exit: 0 bytes in 0 blocks
==3230==   total heap usage: 2 allocs, 2 frees, 40 bytes allocated
==3230==
==3230== All heap blocks were freed -- no leaks are possible
==3230==
==3230== For counts of detected and suppressed errors, rerun with: -v
==3230== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```

3.2.3.4 Lorsque i > n et n = 4

- **Résultat de l'exécution**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
3
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
Voulez vous continuer? 1 = OUI    0 = NON

0
cedric@cedric-X302LA:~/C/TP2SDD$
```

- **Exécution avec valgrind**

```
Que voulez vous faire?

1. Tester les fonctions et procédures de pile.
2. Exécuter la version récursive de TRUC.
3. Exécuter la version itérative de TRUC.
3
Veuillez les éléments à permuter dans le tableau.
Veuillez entrer l'élément 1
1
Veuillez entrer l'élément 2
2
Veuillez entrer l'élément 3
3
Veuillez entrer l'élément 4
4
Voulez vous continuer? 1 = OUI    0 = NON

0
==3331==
==3331== HEAP SUMMARY:
==3331==     in use at exit: 0 bytes in 0 blocks
==3331==   total heap usage: 2 allocs, 2 frees, 40 bytes allocated
==3331==
==3331== All heap blocks were freed -- no leaks are possible
==3331==
==3331== For counts of detected and suppressed errors, rerun with: -v
==3331== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/C/TP2SDD$
```