



ISIMA 1^{ère} ANNEE

Rapport

TP2 de Structure de Données

Anaïs DARRICARRERE
Nada BOUTADGHART

13 mars 2020

Table des matières

1	Présentation générale	1
1.1	Description de l'objet du TP	1
1.2	Description des structures	1
1.2.1	Structure de la pile (LIFO)	1
1.2.2	Strucuture de la file (FIFO)	1
1.3	La fonction TRUC	2
1.4	Organisation du code source	4
1.4.1	Les fichiers d'entête	4
1.4.2	Les modules	5
2	Détails des codes sources de chaque fichier	6
2.1	Pile.h	6
2.2	Pile.c	7
2.3	File.h	12
2.4	File.c	13
2.5	Truc.h	17
2.6	Truc.c	18
2.7	Main.c	22
3	Compte rendu d'exécution	24
3.1	Jeux de test	24
3.2	Makefile	28

Partie 1

Présentation générale

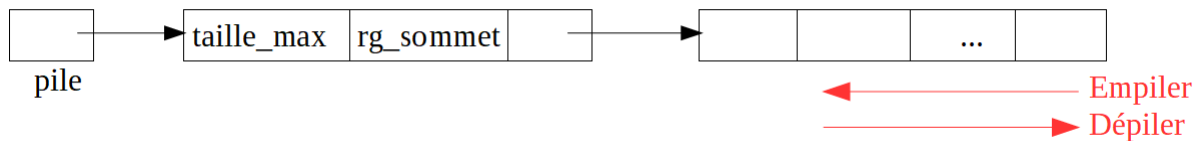
1.1 Description de l'objet du TP

Le but de ce TP est de travailler sur la gestion de la pile et de la file. Par ailleurs, on travaille sur une fonction TRUC qui affiche toutes les permutations possibles des n -i derniers éléments d'un tableau indicé à partir de 0. Cette fonction est écrite de manière récursive puis de manière itérative.

1.2 Description des structures

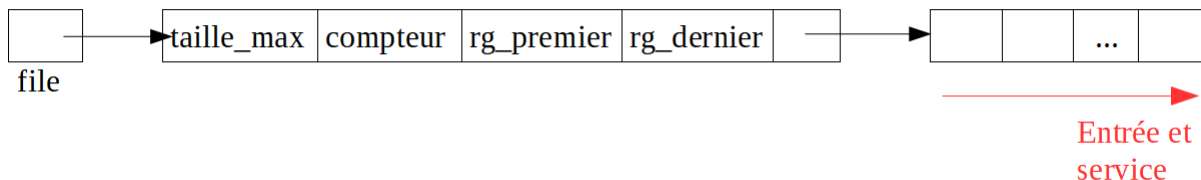
1.2.1 Structure de la pile (LIFO)

La pile fonctionne de la manière suivante : le dernier élément entré dans la pile est le premier à en sortir. Elle a un bloc de tête composé de la taille maximale de la pile, du rang du dernier élément rentré et du pointeur sur une liste contigüe représentant la pile.



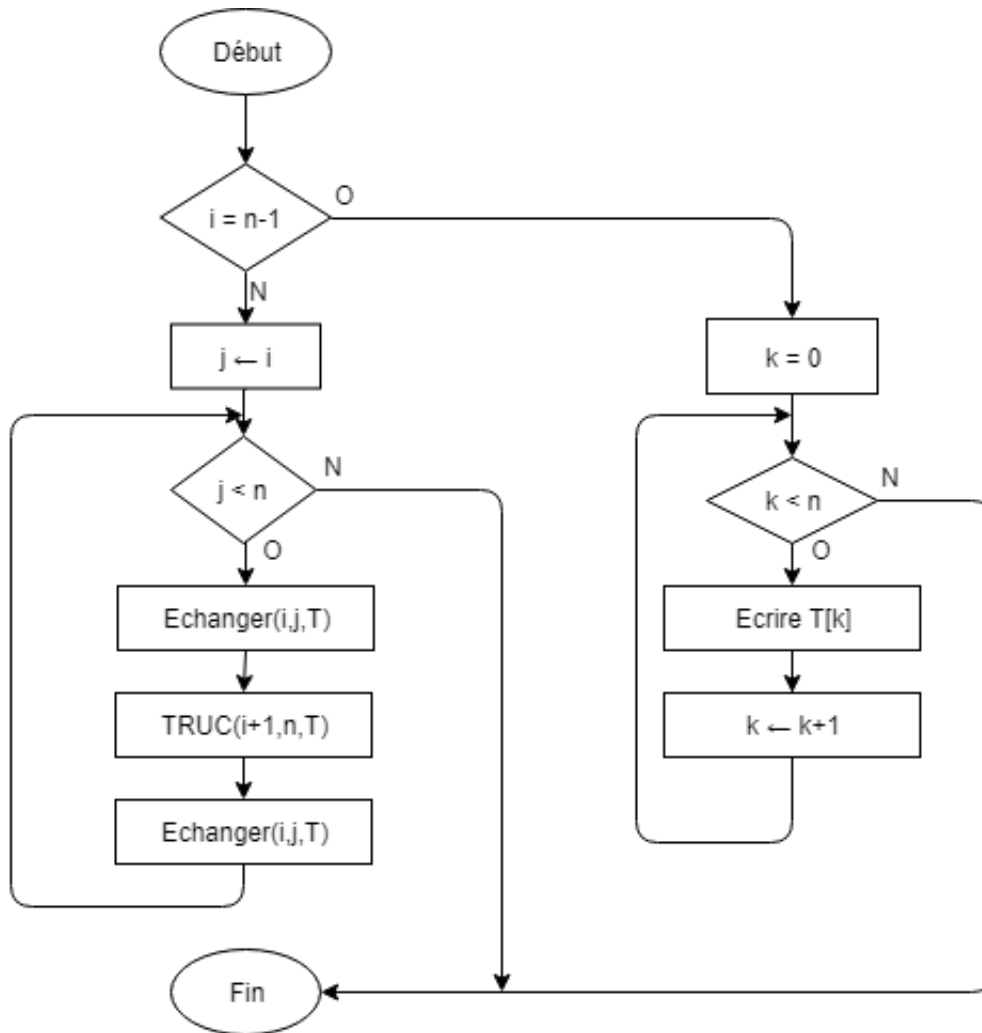
1.2.2 Structure de la file (FIFO)

La file fonctionne de la manière suivante : le premier élément entré dans la file est le premier à en sortir. Elle a un bloc de tête composé de la taille maximale, d'un compteur d'éléments, du rang de l'élément à servir et du rang du dernier élément rentré.



1.3 La fonction TRUC

La fonction TRUC permet d'afficher toutes les permutations des $n-i$ derniers éléments d'un tableau de taille n indicé à partir de 0. Voici l'algorithme de la fonction TRUC récurrente :



Trace sur le tableau : $T = \{3, 9, 2\}$

TRUC(1,3,T) => Pour j de 1 à 3

j = 1

$T = \{3, 9, 2\}$

TRUC(2,3,T) => Pour j de 2 à 3

j = 2

$T = \{3, 9, 2\}$

TRUC(3,3,T) => 3 9 2

$T = \{3, 9, 2\}$

j = 3

$T = \{3, 2, 9\}$

TRUC(3,3,T) => 3 2 9

$T = \{3, 9, 2\}$

$T = \{3, 9, 2\}$

j = 2

$T = \{9, 3, 2\}$

TRUC(2,3,T) => Pour j de 2 à 3

j = 2

$T = \{9, 3, 2\}$

TRUC(3,3,T) => 9 3 2

$T = \{9, 3, 2\}$

j = 3

$T = \{9, 2, 3\}$

TRUC(3,3,T) => 9 2 3

$T = \{9, 3, 2\}$

$T = \{3, 9, 2\}$

J = 3

$T = \{2, 9, 3\}$

TRUC(2,3,T) => Pour j de 2 à 3

j = 2

$T = \{2, 9, 3\}$

TRUC(3,3,T) => 2 9 3

$T = \{2, 9, 3\}$

j = 3

$T = \{2, 3, 9\}$

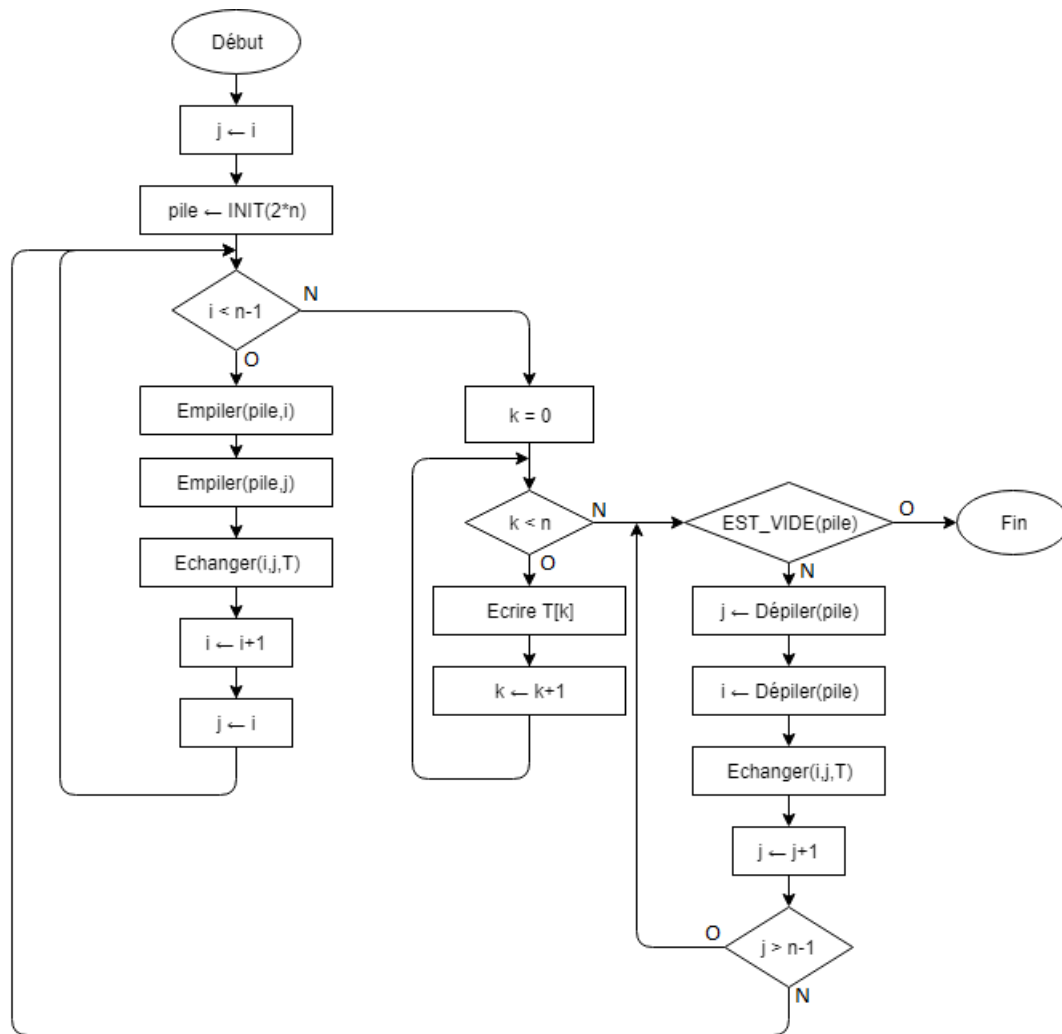
TRUC(3,3,T) => 2 3 9

$T = \{2, 9, 3\}$

$T = \{3, 9, 2\}$

Fin

Voici l'algorithme de la fonction TRUC dérécursiée :



1.4 Organisation du code source

1.4.1 Les fichiers d'entête

Pile.h contient :

- Des directives de préprocesseur permettant d'inclure les bibliothèques `<stdio.h>`, `<stdlib.h>`, `<string.h>` et la déclaration de deux constante, NP représentant la taille de la pile à tester et format représentant le format associé au type `element_pile` sous la forme `"%x"` ;
- La déclaration des types `pile_t` et `element_pile` ;
- Les prototypes des fonctions de gestion de la pile ;

File.h contient :

- Des directives de préprocesseur permettant d'inclure les bibliothèques `<stdio.h>`, `<stdlib.h>`, `<string.h>` et la déclaration de deux constante, NF représentant la taille de la file à tester et format représentant le format associé au type `element_file` sous la forme `"%x"` ;
- La déclaration des types `file_t` et `element_file` ;
- Les prototypes des fonctions de gestion de la file ;

Truc.h contient :

- Des directives de préprocesseur permettant d'inclure les bibliothèques `<stdio.h>`, `<stdlib.h>`, `<string.h>` et `"Pile.h"`,
- Les prototypes de la fonction TRUC écrite de manière récursive et itérative;

1.4.2 Les modules

Pile.c contient :

- Une directive de préprocesseur permettant d'inclure le fichier d'entête `"Pile.h"`;
- Les codes des fonctions de gestion de la pile déclarés dans le fichier `Pile.h` sont :
 - `INIT_PILE(N)` : initialisation d'une pile de taille `N`;
 - `LIBERER_PILE(p)` : libération de la pile;
 - `PILE_VIDE(p)` : teste si la pile est vide;
 - `PILE_PLEINE(p)` : teste si la pile est pleine;
 - `EMPILER(p,element)` : insertion de `element` en tête de la pile;
 - `DEPILER(p)` : suppression de l'élément en tête de la pile;
 - `SOMMET(p)` : retourne l'élément au sommet de la pile;
 - `AFF_PILE(p)` : affichage des éléments contenus dans la pile;

File.c contient :

- Une directive de préprocesseur permettant d'inclure le fichier d'entête `"File.h"`;
- Les codes des fonctions de gestion de la file déclarés dans le fichier `File.h` sont :
 - `INIT_FILE(N)` : initialisation d'une file de taille `N`;
 - `LIBERER_FILE(f)` : libération de la file;
 - `FILE_VIDE(f)` : teste si la file est vide;
 - `FILE_PLEINE(f)` : teste si la file est pleine;
 - `ENTREE(f,element)` : insertion de `element` en fin de la file;
 - `SORTIE(f)` : suppression de l'élément en tête de la file;
 - `AFF_FILE(p)` : affichage des éléments contenus dans la file

Truc.c contient :

- Une directive de préprocesseur permettant d'inclure le fichier d'entête `"Truc.h"`;
- Les codes des fonctions récursives et itératives de la fonction TRUC déclarés dans le fichier `Truc.h` sont :
 - `TRUC_REC(i,n,T)` : affichage de toutes les permutations des `n-i` derniers éléments d'un tableau `T` de taille `n` de manière récursive;
 - `ECHANGE(i,j,T)` : échange les éléments d'indices `i` et `j` du tableau `T`;
 - `TRUC_ITER(i,n,T)` : affichage de toutes les permutations des `n-i` derniers éléments d'un tableau `T` de taille `n` de manière itérative;

Main.c contient :

- Des directives de préprocesseur permettant d'inclure les fichiers d'entête `"Truc.h"` et `"File.h"`;
- Le code du programme principal qui teste les fonctions de gestion de la pile, de la file et qui teste les deux fonctions TRUC;

Partie 2

Détails des codes sources de chaque fichier

2.1 Pile.h

```
/*-----*/
/*                      TP2 - Gestion d'une pile et d'une file                      */
/*                      Fichier d'entête Pile.h                                    */
/*                      */
/* Déclaration des structures et des fonctions                                    */
/*-----*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NP 3                      /* taille de la pile */

/*-----*/
/* Déclaration du type d'element contenu dans la pile                            */
/*-----*/

#define format "%d"
typedef int element_pile ;

/*-----*/
/* Déclaration de la structure pile avec bloc de tête                            */
/*      taille_max      taille de la pile                                        */
/*      rg_sommet       rang du sommet de la pile                              */
/*      ad_pile         pointeur sur la pile                                    */
/*-----*/

typedef struct pile
{
    int taille_max, rg_sommet;
    element_pile * ad_pile;
}pile_t;

/*-----*/
/* Déclaration des fonctions                                                    */
/*-----*/

pile_t * INIT_PILE (int N);

void LIBERER_PILE(pile_t * p);

int PILE_VIDE(pile_t * p);

int PILE_PLEINE(pile_t * p);

int EMPILER(pile_t* p, element_pile element);

element_pile DEPILER(pile_t *p);

element_pile SOMMET(pile_t *p);

void AFF_PILE(pile_t * p);
```


2.2 Pile.c

```
/*-----*/
/*          TP2 - Gestion d'une pile et d'une file          */
/*                      Pile.c                      */
/*-----*/

#include "Pile.h"

/*-----*/
/* INIT_PILE          Initialisation d'une pile          */
/*                      */
/* Algorithme de principe                      */
/* - Allouer le bloc de tête de la pile          */
/* - Si l'allocation est réussie:                */
/*     - on initialise les éléments du bloc de tête */
/*     - on alloue le bloc pile                    */
/*     - Si l'allocation n'est pas réussie, on libère le */
/*         bloc de tête                            */
/*                      FIN                      */
/*                      */
/* Lexique                      */
/*                      */
/* En entrée : N                Taille de la pile          */
/*                      */
/* En sortie : p                Pointeur sur le bloc de tête de la pile */
/*-----*/

pile_t * INIT_PILE (int N)
{
    pile_t * p;
    p = (pile_t*)malloc(sizeof(pile_t));
    if (p != NULL)
    {
        p->taille_max = N;
        p->rg_sommet = -1;
        p->ad_pile = (element_pile *)malloc(sizeof(element_pile)*N);
        if (p->ad_pile == NULL)
        {
            free(p);
            p = NULL;
        }
    }
    return p;
}
```

```

/*-----*/
/* LIBERER_PILE          Libération d'une pile          */
/*                                                              */
/* Algorithme de principe                                     */
/*      - Libérer la pile                                   */
/*      - Libérer le bloc de tête de la pile              */
/*      FIN                                                 */
/* Lexique                                                  */
/* En entrée : p      Pointeur sur le bloc de tête de la pile */
/*-----*/

void LIBERER_PILE(pile_t * p)
{
    free(p->ad_pile);
    free(p);
}

/*-----*/
/* PILE_VIDE             Détermine si la pile est vide     */
/*                                                              */
/* Lexique                                                      */
/* En entrée : p      Pointeur sur le bloc de tête de la pile */
/* En sortie :         Retourne 1 si la pile est vide et 0 sinon */
/*-----*/

int PILE_VIDE(pile_t * p)
{
    return (p->rg_sommet == -1);
}

/*-----*/
/* PILE_PLEINE           Détermine si la pile est pleine   */
/*                                                              */
/* Lexique                                                      */
/* En entrée : p      Pointeur sur le bloc de tête de la pile */
/* En sortie :         Retourne 1 si la pile est pleine et 0 sinon */
/*-----*/

int PILE_PLEINE(pile_t * p)
{
    return (p->rg_sommet == p->taille_max -1);
}

```

```

/*-----*/
/* EMPILER          Ajoute un élément en tête de la pile          */
/*                                                         */
/* Algorithme de principe                                         */
/*      - Si la pile n'est pas pleine :                           */
/*      - on incrémente le rang du premier élément              */
/*      - on insère l'élément                                     */
/*      FIN                                                         */
/*                                                         */
/* Lexique                                                         */
/* En entrée : p          Pointeur sur le bloc de tête de la pile */
/*      element          Élément à empiler                        */
/*                                                         */
/* En sortie : ok         Retourne 1 si l'élément est empilé      */
/*-----*/

int EMPILER(pile_t* p, element_pile element)
{
    int ok;
    ok = !(PILE_PLEINE(p));
    if (ok)
    {
        p->rg_sommet = (p->rg_sommet)+1;
        *(p->ad_pile + p->rg_sommet) = element;
    }
    else
        printf("Erreur : pile pleine\n");
    return ok;
}

```

```

/*-----*/
/* DEPILER          Enleve l'élément au sommet de la pile          */
/*                                                         */
/* Algorithme de principe                                     */
/*      - Si la pile n'est pas vide:                         */
/*      - on récupère l'élément à dépiler                   */
/*      - on décrémente le rang du premier élément          */
/*      FIN                                                  */
/*                                                         */
/* Lexique                                                  */
/* En entrée : p      Pointeur sur le bloc de tête de la pile */
/* Variables intermédiaires :                               */
/*      ok            Entier valant 0 si la pile est vide et 1 sinon */
/*                                                         */
/* En sortie : res     Retourne l'élément à enlever de la pile */
/*-----*/

```

```

element_pile DEPILER(pile_t *p)
{
    int ok;
    element_pile res;
    ok = !(PILE_VIDE(p));
    if (ok)
    {
        res = *(p->ad_pile + p->rg_sommet);
        p->rg_sommet = (p->rg_sommet)-1;
    }
    else
        printf("Erreur : pile vide\n");
    return res;
}

```

```

/*-----*/
/* SOMMET           Retourne l'élément au sommet de la pile       */
/*                                                         */
/* Algorithme de principe                                     */
/*      - Si la pile n'est pas vide:                         */
/*      - on récupère l'élément au sommet                   */
/*      FIN                                                  */
/*                                                         */
/* Lexique                                                  */
/* En entrée : p      Pointeur sur le bloc de tête de la pile */
/* Variables intermédiaires :                               */
/*      ok            Entier valant 0 si la pile est vide et 1 sinon */
/*                                                         */
/* En sortie : res     Retourne l'élément à enlever de la pile */
/*-----*/

```

```

element_pile SOMMET(pile_t *p)
{
    int ok;
    element_pile res = -1;
    ok = !(PILE_VIDE(p));
    if (ok)
    {
        res = *(p->ad_pile + p->rg_sommet);
    }
    return res;
}

```

```

/*-----*/
/* AFF_PILE          Affiche la liste des éléments contenus dans la pile */
/* */
/* Algorithme de principe */
/* - Si la pile n'est pas vide: */
/*   - On parcourt la pile de l'élément en tête à celui */
/*     en fin en affichant chacun de ces éléments */
/* - Sinon : afficher un message d'erreur */
/* FIN */
/* */
/* Lexique */
/* */
/* En entrée : p      Pointeur sur le bloc de tête de la pile */
/* */
/* Variables intermédiaires : */
/*     nb_elem        entier indiquant le nombre d'éléments contenus dans la pile */
/* */
/*-----*/

void AFF_PILE(pile_t * p)
{
    int nb_elem = p->rg_sommet +1;
    int i;
    if (nb_elem != 0)
    {
        element_pile * deb = (p->ad_pile) + (p->rg_sommet);
        printf("[ ");
        for(i=0; i<nb_elem; i++)
        {
            printf(format,*(deb-i));
            printf(" ");
        }
        printf("]\n");
    }
    else
        printf("Pile vide\n");
}

```

2.3 File.h

```
/*-----*/
/*          TP2 - Gestion d'une pile et d'une file          */
/*          Fichier d'entête File.h                        */
/*-----*/
/* Déclaration des structures et des fonctions              */
/*-----*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NF 3

/*-----*/
/* Déclaration du type d'element contenu dans la file      */
/*-----*/

#define format "%d"
typedef int element_file ;

/*-----*/
/* Déclaration de la structure pile avec bloc de tête      */
/*      taille_max      taille de la pile                  */
/*      rg_sommet       rang du sommet de la pile          */
/*      ad_pile         pointeur sur la pile                */
/*-----*/

typedef struct file
{
    int taille_max, compteur, rg_premier, rg_dernier;
    element_file * ad_file;
}file_t;

/*-----*/
/* Déclaration des fonctions de gestion de la file        */
/*-----*/

file_t * INIT_FILE (int N);

void LIBERER_FILE(file_t * f);

int FILE_VIDE(file_t * f);

int FILE_PLEINE(file_t * f);

int ENTREE(file_t * f, element_file element);

int SORTIE(file_t * f);

void AFF_FILE(file_t * f);
```

2.4 File.c

```
/*-----*/
/*          TP1 - Gestion d'une pile et d'une file          */
/*          Module File.c          */
/*-----*/

#include "File.h"

/*-----*/
/* INIT_FILE          Initialisation d'une file          */
/*          */
/* Algorithme de principe          */
/*          - Allouer le bloc de tête de la file          */
/*          - Si l'allocation est réussie :          */
/*              - Allouer le bloc file          */
/*              - Si l'allocation est réussie :          */
/*                  - Initialiser les éléments du bloc de tête          */
/*              - Sinon : Libérer le bloc de tête de la file          */
/*          - Sinon : Afficher un message d'erreur          */
/*          FIN          */
/*          */
/* Lexique          */
/*          */
/* En entrée : N          Taille de la pile          */
/*          */
/* En sortie : f          Pointeur sur le bloc de tête de la file          */
/*-----*/

file_t * INIT_FILE (int N)
{
    file_t * f;
    f = (file_t*)malloc(sizeof(file_t));
    if (f != NULL)
    {
        f->ad_file = (element_file *)malloc(N*sizeof(element_file));
        if (f->ad_file != NULL)
        {
            f->taille_max = N;
            f->compteur = 0;
            f->rg_premier = 0;
            f->rg_dernier = N-1;
        }
        else
        {
            free(f);
            f = NULL;
        }
    }
    else
    {
        printf("Erreur : probleme d'allocation\n");
    }
    return f;
}
```

```

/*-----*/
/* LIBERER_FILE          Libération d'une file          */
/*                      */
/* Lexique              */
/*                      */
/* En entrée : f        Pointeur sur le bloc de tête de la pile */
/*-----*/

void LIBERER_FILE(file_t * f)
{
    free(f->ad_file);
    free(f);
}

/*-----*/
/* FILE_VIDE             Détermine si la file est vide   */
/*                      */
/* Lexique              */
/*                      */
/* En entrée : f        Pointeur sur le bloc de tête de la file */
/*                      */
/* En sortie :          Retourne 1 si la file est vide et 0 sinon */
/*-----*/

int FILE_VIDE(file_t * f)
{
    return (f->compteur == 0);
}

/*-----*/
/* FILE_PLEINE           Détermine si la file est pleine */
/*                      */
/* Lexique              */
/*                      */
/* En entrée : f        Pointeur sur le bloc de tête de la file */
/*                      */
/* En sortie :          Retourne 1 si la file est pleine et 0 sinon */
/*-----*/

int FILE_PLEINE(file_t * f)
{
    return (f->compteur == f->taille_max);
}

```



```

/*-----*/
/* ENTREE          Ajoute un élément en fin de la file */
/* */
/* Algorithme de principe */
/* - Si la file n'est pas pleine : */
/*     - on incrémente le rang du dernier élément */
/*     de la file (modulo la taille max de la file) */
/*     - on copie l'élément à insérer en fin de file */
/*     - on incrémente le compteur d'éléments de la file */
/* - Sinon : afficher un message d'erreur */
/*     FIN */
/* */
/* Lexique */
/* */
/* En entrée : f      Pointeur sur le bloc de tête de la file */
/*     element      Elément à insérer dans la file */
/* */
/* Variables intermédiaires : */
/*     taille      taille maximale de la file */
/* */
/* En sortie : ok      Retourne 1 si l'élément est inséré, 0 si la file est */
/*     pleine et l'élément n'a pas pu être inséré */
/*-----*/

```

```

int ENTREE(file_t* f, element_file element)

```

```

{
    int taille = f->taille_max;
    int ok;
    ok = !(FILE_PLEINE(f));
    if (ok)
    {
        f->rg_dernier = (f->rg_dernier+1)%taille;
        *(f->ad_file + f->rg_dernier) = element;
        f->compteur++;
    }
    else
    {
        printf("Erreur : File pleine\n");
    }
    return ok;
}

```

```

/*-----*/
/* SORTIE          Retourne et supprime l'élément en tête de file */
/* */
/* Algorithme de principe */
/* - Si la file n'est pas vide: */
/*     - on récupère l'élément à servir en tête de file */
/*     - on incrémente le rang du premier élément */
/*       (modulo la taille max de la file) */
/*     - on décrémente le compteur d'éléments de la file */
/* - Sinon : afficher un message d'erreur */
/* FIN */
/* */
/* Lexique */
/* */
/* En entrée : f      Pointeur sur le bloc de tête de la file */
/* */
/* Variables intermédiaires : */
/*     taille      entier indiquant la taille maximale de la file */
/*     ok          Entier valant 0 si la file est vide et 1 sinon */
/* */
/* En sortie : res     Retourne l'élément à sortir de la file */
/*-----*/

```

```

element_file SORTIE(file_t *f)

```

```

{
    int taille = f->taille_max;
    element_file res;
    int ok;
    ok = !(FILE_VIDE(f));
    if (ok)
    {
        res = *(f->ad_file + f->rg_premier);
        f->rg_premier = (f->rg_premier+1)%taille;
        f->compteur--;
    }
    else
        printf("Erreur : File vide\n");
    return res;
}

```

```

/*-----*/
/* AFF_FILE          Affiche la liste des éléments contenus dans la file */
/*                                                           */
/* Algorithme de principe */
/* - Si la file n'est pas vide: */
/*     - On parcourt la file de l'élément en tête à celui */
/*       en fin en affichant chacun de ces éléments */
/*     - Sinon : afficher un message d'erreur */
/*     FIN */
/*                                                           */
/* Lexique */
/* En entrée : f      Pointeur sur le bloc de tête de la file */
/* Variables intermédiaires : */
/*     nb_elem        entier indiquant le nombre d'éléments contenus dans la file */
/*-----*/

void AFF_FILE(file_t * f)
{
    int nb_elem = f->compteur;
    int i;
    if (nb_elem != 0)
    {
        element_file * deb = (f->ad_file) + (f->rg_premier);
        printf("[ ");
        for(i=0; i<nb_elem; i++)
        {
            printf(format,*(deb+i));
            printf(" ");
        }
        printf("]\n");
    }
    else
        printf("File vide\n");
}

```

2.5 Truc.h

```

/*-----*/
/* TP2 - Gestion d'une pile et d'une file */
/* Fichier d'entête Truc.h */
/*                                                           */
/* Déclaration des fonctions */
/*-----*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Pile.h"

void TRUC_REC(int i, int n, element_pile *T);

void ECHANGE(int i, int j, element_pile * T);

void TRUC_ITER(int i, int n, element_pile * T);

```

2.6 Truc.c

```
/*-----*/
/*          TP2 - Gestion d'une pile et d'une file          */
/*          Truc.c                                          */
/*-----*/

#include "Truc.h"

/*-----*/
/* TRUC_REC          Retourne toutes les permutations n-i+1 derniers éléments */
/*                  d'un tableau initial de manière récursive                */
/*                  */
/* Algorithme de principe */
/* - Si l'indice i est supérieur à la taille du tableau: on */
/* retourne un message d'erreur */
/* - Sinon */
/* - Si l'indice i est égal à la taille du tableau: on */
/* affiche le tableau */
/* - Sinon : */
/* - Pour j allant de i à n : */
/* - On échange les i-ème et j-ème éléments du */
/* tableau */
/* - On affiche le tableau */
/* - On rééchange les i-ème et j-ème éléments du */
/* tableau */
/* */
/* Lexique */
/* */
/* En entrée : i      Indice à partir duquel on commence les permutations */
/*              n      Taille du tableau */
/*              T      Tableau donnée en entrée indicé à partir de 0 */
/*-----*/
```

```
void TRUC_REC(int i, int n, element_pile *T)
```

```
{
    int j;
    element_pile temp;
    if (i > n-1)
    {
        printf("Erreur: l'indice i est superieur a la taille du tableau");
    }
    else
    {
        if (i == n-1)
        {
            for (j=0; j<n; j++)
                printf(format, T[j]);
            printf("\n");
        }
        else
        {
            for (j=i; j<n; j++)
            {
                temp = T[i];
                T[i] = T[j];
                T[j] = temp;
                TRUC_REC(i+1,n,T);
                temp = T[i];
                T[i] = T[j];
                T[j] = temp;
            }
        }
    }
}
```

```
/*-----*/
/* ECHANGE          Echange deux éléments dans un tableau          */
/*                                                         */
/* Algorithme de principe                                                         */
/*      - On crée une variable temporaire pour sauvegarder          */
/*      l'élément d'indice i                                          */
/*      - On remplace l'élément d'indice i par celui d'indice j    */
/*      - On remplace l'élément d'indice j par celui sauvegarder   */
/*      dans temp                                                    */
/*                                                         */
/* Lexique                                                         */
/*                                                         */
/* En entrée : i,j          Indices des deux éléments à changer     */
/*              T           Tableau donnée en entrée indicé à partir de 0 */
/*-----*/
```

```
void ECHANGE(int i, int j, element_pile * T)
```

```
{
    int temp;
    temp = T[i];
    T[i] = T[j];
    T[j] = temp;
}
```

```

/*-----*/
/* TRUC_ITER          Retourne toutes les permutations n-i+1 derniers éléments */
/*                   d'un tableau initial de manière itérative */
/*                   */
/* Algorithme de principe */
/* - On initialise la pile */
/* - Tant que la variable d'arrêt est nulle: */
/*   - Si l'indice i est inférieur à la taille du tableau -1 */
/*     - On empile l'élément i puis l'élément j */
/*     - On échange les éléments d'indice i et j */
/*     - On incrémente i */
/*     - On affecte i à j */
/*   - Sinon (soit i = n): */
/*     - On affiche le tableau */
/*     - Répéter: */
/*       - Si la variable d'arrêt est nulle : */
/*         - On dépile en affectant à j l'élément du */
/*           sommet */
/*         - On dépile en affectant à i l'élément du */
/*           sommet */
/*         - On échange les éléments d'indice i et j */
/*         - On incrémente j */
/*       - Sinon: on affecte 1 à la variable d'arrêt */
/*     Jusqu'à ce que j soit inférieur à n-1 et que la */
/*     variable d'arrêt soit égale à 1 */
/* Lexique */
/* En entrée : i      Indice à partir duquel on commence les permutations */
/*              n      Taille du tableau */
/*              T      Tableau donnée en entrée indicé à partir de 0 */
/* Variables temporaires: */
/* k,j            Indices pour parcourir le tableau */
/* stop          Entier qui arrête le programme si elle vaut 1 et 0 sinon */
/* pile          Pointeur sur le bloc de tête de la pile
/*-----*/

```

```

void TRUC_ITER(int i, int n, element_pile * T)
{
    int k, j = i, stop = 0;
    pile_t * pile = NULL;
    pile = INIT_PILE(2*n);
    while (!stop)
    {
        if (i<n-1)
        {
            EMPILER(pile, i);
            EMPILER(pile, j);
            ECHANGE(i,j,T);
            i++;
            j = i;
        }
        else
        {
            for (k = 0; k<n; k++)
                printf(format, T[k]);
            printf("\n");
            do
            {
                if(!PILE_VIDE(pile))
                {
                    j = DEPIILER(pile);
                    i = DEPIILER(pile);
                    ECHANGE(i,j,T);
                    j++;
                }
                else
                    stop = 1;
            }while((!stop) && j>n-1);
        }
    }
}

```

2.7 Main.c

```
/*-----*/
/*          TP2 - Gestion d'une pile et d'une file          */
/*                               Main.c                               */
/*-----*/

#include "File.h"
#include "Truc.h"

int main()
{
    int i;
    int T[3] = {3,9,2};

    // Test de la gestion de la pile
    printf("-Test de la gestion de la pile\n");
    printf("Creation d'une pile vide de taille 3 :\n");
    pile_t * p;
    p = INIT_PILE(NP);
    AFF_PILE(p);

    printf("Empilement de 3 éléments dans la pile :\n");
    EMPILER(p,6);
    AFF_PILE(p);
    EMPILER(p,7);
    AFF_PILE(p);
    EMPILER(p,8);
    AFF_PILE(p);

    printf("Empilement d'un element dans la pile pleine :\n");
    EMPILER(p,9);

    printf("Depilement des 3 elements dans la pile :\n");
    for (i=0; i<NP; i++)
    {
        printf("Element depile : \"format\" Pile :\", DEPILER(p));
        AFF_PILE(p);
    }
    printf("Depilement dans la pile vide :\n");
    DEPILER(p);

    printf("\n");
}
```



```

// Test de la gestion de la file
printf("-Test de la gestion de la file\n");
printf("Creation d'une pile vide de taille 3 : ");
file_t * f;
f = INIT_FILE(NF);
AFF_FILE(f);

printf("Entree de 3 elements en fin de la file :\n");
ENTREE(f,5);
AFF_FILE(f);
ENTREE(f,3);
AFF_FILE(f);
ENTREE(f,7);
AFF_FILE(f);

printf("Entree d'un elements dans la file pleine : \n");
ENTREE(f,1);

printf("Services successifs des 3 elements dans la file : \n");
for(i=0; i<NF; i++)
{
    printf("Element servi : \"format\" File :", SORTIE(f));
    AFF_FILE(f);
}
printf("Service dans la file vide :\n");
SORTIE(f);

printf("\n");

// Test de la fonction TRUC réursive
printf("-Test de la fonction TRUC réursive :\n");
TRUC_REC(0,3,T);

printf("\n");

// Test de la fonction TRUC itérative
printf("-Test de la fonction TRUC itérative :\n");
TRUC_ITER(0,3,T);
}

```

Partie 3

Compte rendu d'exécution

3.1 Jeux de test

Les tests des fonctions TRUC récursive et itérative ont été effectués sur le tableau à 3 éléments suivant :

$$\mathbf{T} = \{3,9,2\}$$

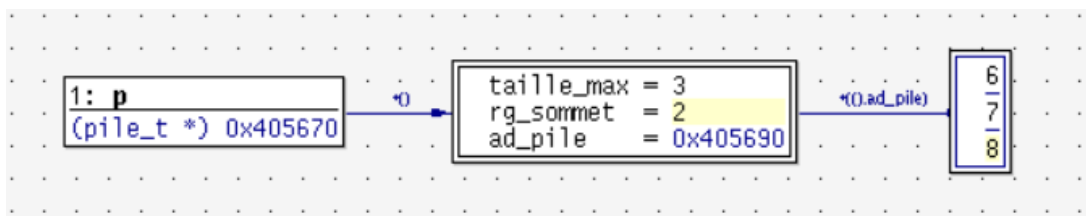
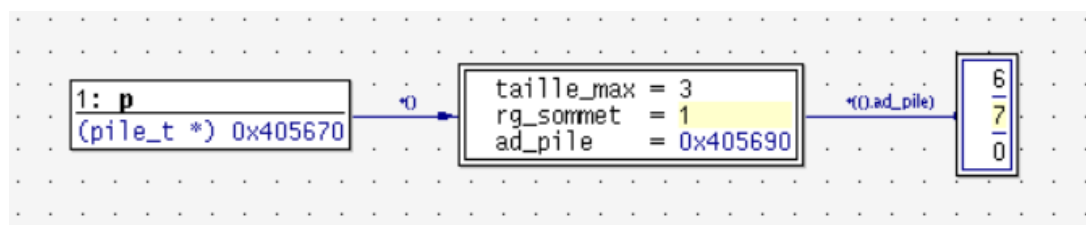
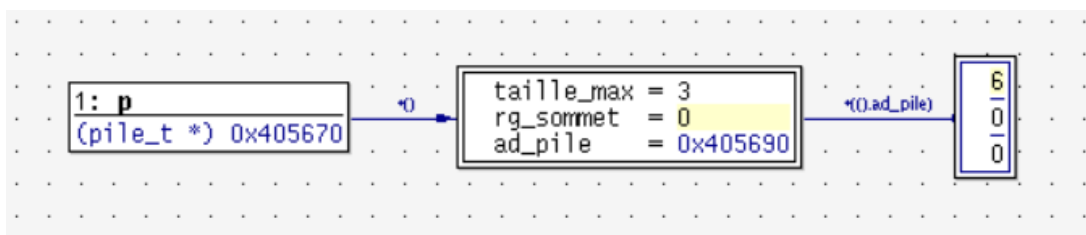
Les résultats des différents tests sont observés à travers le terminal et le débogueur ddd pour l'affichage des piles et des files.

- Création d'une pile vide
- Empiler dans une pile non pleine et affichage du sommet
- Empiler dans une pile pleine et affichage du sommet
- Dépiler dans une pile non vide
- Dépiler dans une pile vide
- Création d'une file vide
- Insertion en fin dans une file non pleine
- Insertion en fin dans une file pleine
- Service dans une file vide
- Service dans une file non vide
- Test de la fonction TRUC récursive
- Test de la fonction TRUC itérative

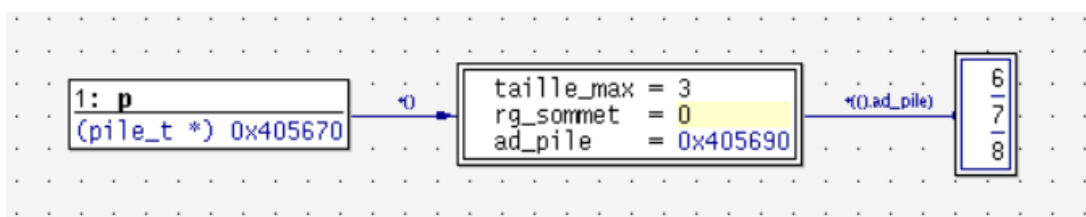
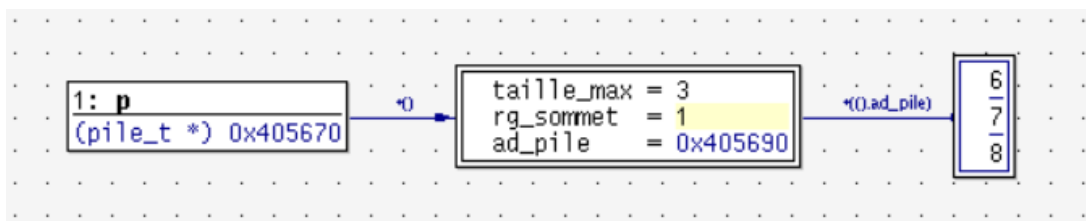
Les résultats des tests effectués sont les affichés ci-après :

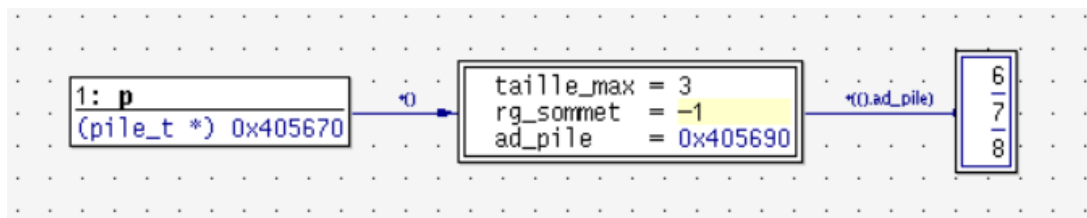
— Test des fonctions de gestion de pile :

```
[naboutadgh@etud TP2SDD]$ ./TP2
-Test de la gestion de la pile
Creation d'une pile vide de taille 3 :
Pile vide
Empilement de 3 éléments dans la pile :
[ 6 ]
[ 7 6 ]
[ 8 7 6 ]
Empilement d'un element dans la pile pleine :
Erreur : pile pleine
```



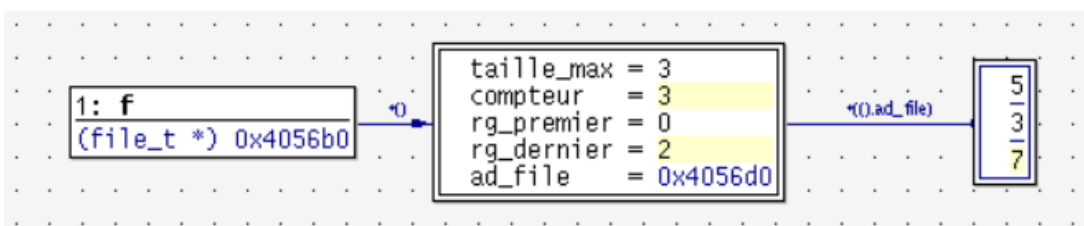
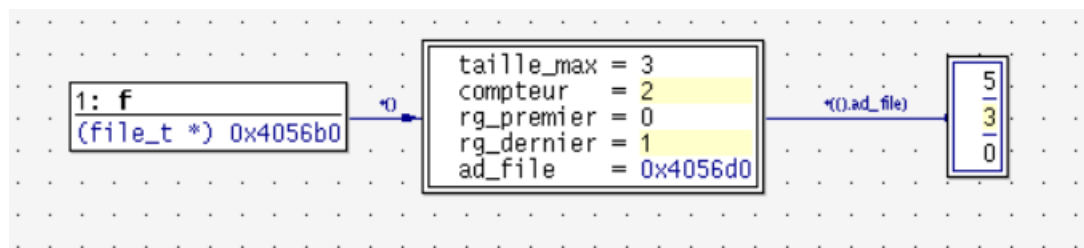
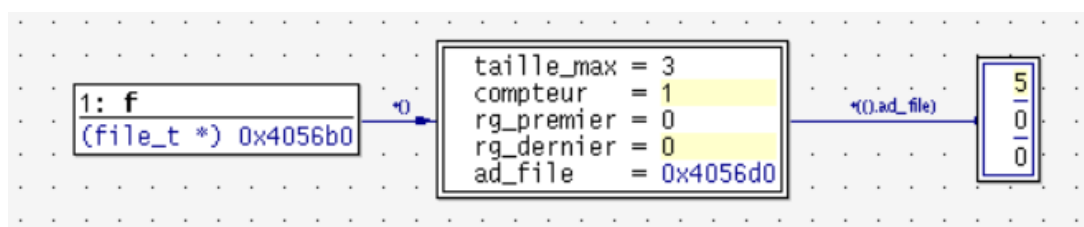
```
Depilement des 3 elements dans la pile :
Element depile : 8 Pile : [ 7 6 ]
Element depile : 7 Pile : [ 6 ]
Element depile : 6 Pile : Pile vide
Depilement dans la pile vide :
Erreur : pile vide
```



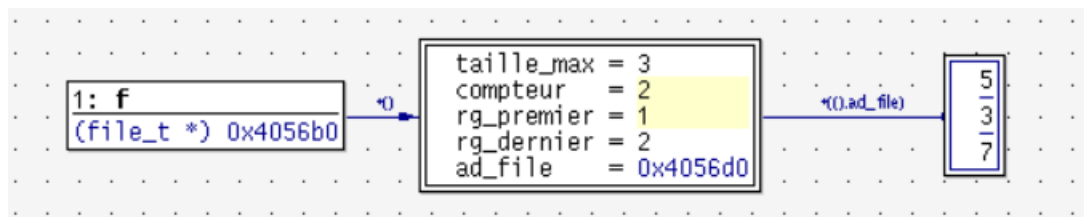


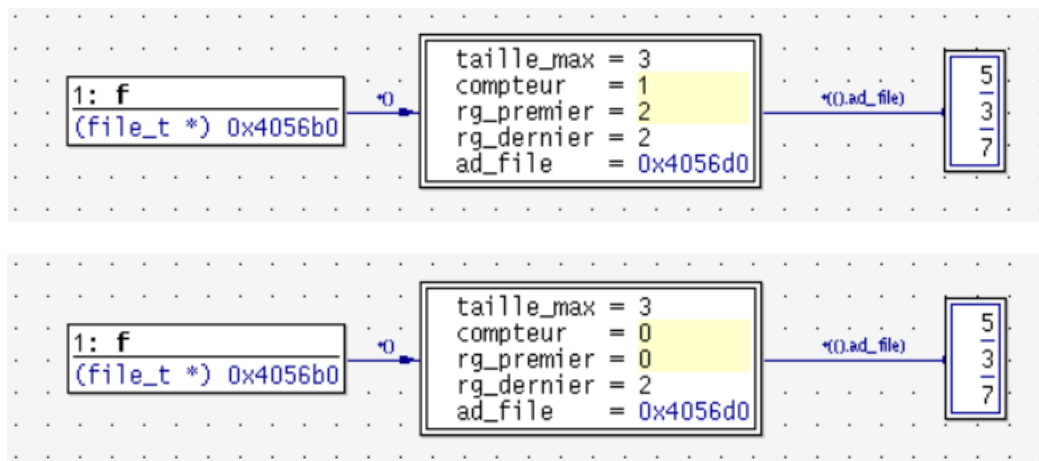
— Test des fonctions de gestion de file :

```
-Test de la gestion de la file
Creation d'une pile vide de taille 3 : File vide
Entree de 3 elements en fin de la file :
[ 5 ]
[ 5 3 ]
[ 5 3 7 ]
Entree d'un elements dans la file pleine :
Erreur : File pleine
```



```
Services successifs des 3 elements dans la file :
Element servi : 5 File :[ 3 7 ]
Element servi : 3 File :[ 7 ]
Element servi : 7 File :File vide
Service dans la file vide :
Erreur : File vide
```





— Test des fonctions TRUC récursive et itérative :

```
-Test de la fonction TRUC récursive :
392
329
932
923
293
239

-Test de la fonction TRUC itérative :
392
329
932
923
293
239
```

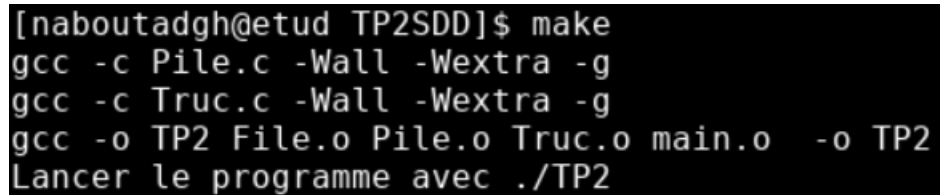
3.2 Makefile

```
# compilateur
CC = gcc
# options
CFLAGS = -Wall -Wextra -g
LDFLAGS =
# liste des fichiers objets
OBJ = File.o Pile.o Truc.o main.o
# règle de production finale tp :

TP2 : $(OBJ)
    $(CC) -o TP2 $(OBJ) $(LDFLAGS) -o TP2
    @echo "Lancer le programme avec ./TP2"

# règle de production pour chaque fichier
File.o : File.h File.c
    $(CC) -c File.c $(CFLAGS)
Pile.o : Pile.h Pile.c
    $(CC) -c Pile.c $(CFLAGS)
Truc.o : Truc.h Pile.h Truc.c
    $(CC) -c Truc.c $(CFLAGS)
```

À la compilation, la consigne suivante s'affiche sur le terminal :

A terminal window with a black background and white text. The prompt is [naboutadgh@etud TP2SDD]\$ and the command entered is make. The output shows three compilation commands: gcc -c Pile.c -Wall -Wextra -g, gcc -c Truc.c -Wall -Wextra -g, and gcc -o TP2 File.o Pile.o Truc.o main.o -o TP2. The final line of output is Lancer le programme avec ./TP2.

```
[naboutadgh@etud TP2SDD]$ make
gcc -c Pile.c -Wall -Wextra -g
gcc -c Truc.c -Wall -Wextra -g
gcc -o TP2 File.o Pile.o Truc.o main.o -o TP2
Lancer le programme avec ./TP2
```