TP2
Pile, File et Dérécursification

Mathieu ARQUILLIERE

Table des matières

I – Présentation

L'objectif de ce TP est de créer 2 bibliothèques indépendantes, une pour la gestion d'une pile et l'autre pour la gestion d'une file. Il faut également dérécursifier une fonction donnée (en utilisant l'implémentation de la pile faite).

II – Description des Structures de données

1) La pile

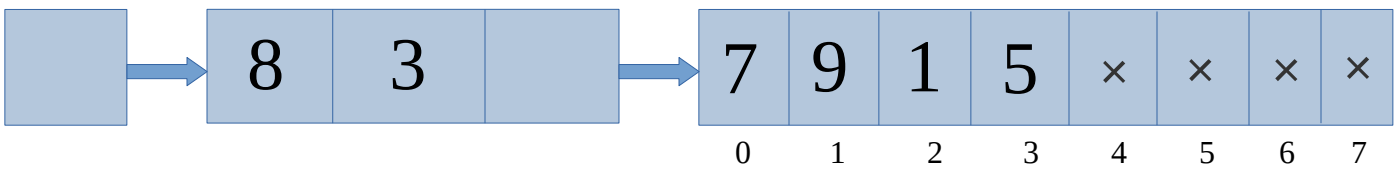
La pile est constituée d'une tête contenant :

- La taille maximale : entier représentant le nombre maximum d'élément que peut contenir la pile.
- Le rang du sommet : entier représentant l'adresse dans le tableau associé à la pile l'élément au dessus de la pile.
- Le tableau associé à la pile : tableau contenant les éléments de la pile.

La pile est utilisée via les fonctions / procédures associées :

- INIT : initialisation de la pile (tête et tableau associé).
- EST_VIDE : teste si la pile est vide.
- EST_PLEINE : teste si la pile est pleine.
- EMPILER : ajoute un élément sur le dessus de la pile.
- DEPILER : enlève et renvoie l'élément sur le dessus de la pile.
- SOMMET : renvoie l'élément sur le dessus de la pile.
- LIBERER : libère l'espace mémoire du tableau et de la tête de la pile.

Exemple de pile :



2) La File

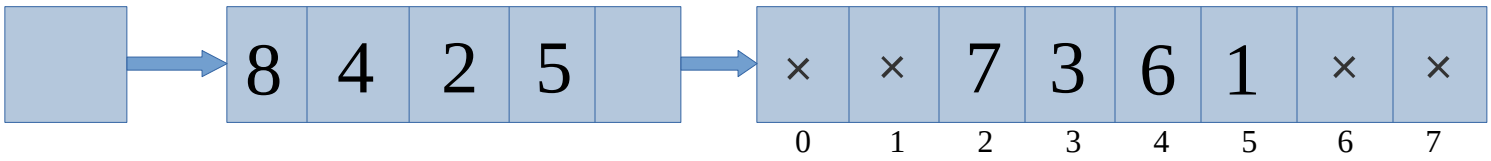
La file est constituée d'une tête contenant :

- La taille maximale : entier représentant le nombre maximum d'élément que peut contenir la file.
- Un compteur : entier qui compte le nombre d'éléments dans la file.
- Le rang du premier : entier représentant l'adresse dans le tableau associé à la file l'élément au début de la file (le prochain à défiler).
- Le rang du dernier : entier représentant l'adresse dans le tableau associé à la file l'élément à la fin de la file (le dernier arrivé).
- Le tableau associé à la file : tableau contenant les éléments de la file.

La pile est utilisée via les fonctions / procédures associées :

- INIT : initialisation de la file (tête et tableau associé).
- EST_VIDE : teste si la file est vide.
- EST_PLEINE : teste si la file est pleine.
- ENFILER : ajoute un élément à la fin de la file.
- DEFILER : enlève et renvoie l'élément au début de la file.
- LIBERER : libère l'espace mémoire du tableau et de la tête de la file.

Exemple de file :



III – Organisation du code

1) Module Pile (pile.c / pile.h)

```
struct Pile
{
    int tailleMax;
    int rangSommet;
    T   tab;
};
typedef struct Pile Pile_t;
```

La pile est représentée par une structure contenant un entier pour la taille maximale, un entier pour le rang sommet et un tableau d' éléments. Le type d'élément est défini auparavant grâce à un typedef (ainsi on peut utiliser le même module pour différents types en changeant uniquement ce typedef).

```
typedef int T;
```

Le module contient les fonctions associées à la pile :

```
int     initPile    (Pile_t** pile, int taille);
int     pileVide    (Pile_t* pile);
int     pilePleine  (Pile_t* pile);
int     empiler     (Pile_t* pile, T element);
int     depiler     (Pile_t* pile, T* element);
int     sommet      (Pile_t* pile, T* element);
void    libererPile (Pile_t* pile);
void    debugPile   (Pile_t* pile);
```

initPile : Allocation et initialisation de la tête de la pile, allocation du tableau de la pile.

pileVide : Teste si la pile est vide ou non.

pilePleine : Teste si la pile est pleine ou non.

empiler : Ajoute un élément sur le dessus de la pile.

depiler : Retire et renvoie l'élément sur le dessus de la pile.

sommet : Renvoie l'élément sur le dessus de la pile.

libererPile : Libère la mémoire du tableau et de la tête de la pile.

debugPile : Affiche le contenu de la tête et du tableau de la pile.

2) Module File (file.c / file.h)

```
struct File
{
    int tailleMax;
    int compteur;
    int rangPremier;
    int rangDernier;
    T*  tab;
};
typedef struct File File_t;
```

La file est représentée par une structure contenant un entier pour la taille maximale, un entier pour le nombre d'éléments, un entier pour le rang de l'élément en début de file, un entier pour le rang de l'élément en fin de file et un tableau d' éléments. Le type d'élément est défini auparavant grâce à un typedef (de la même façon de la file).

```
typedef int T;
```

Le module contient les fonctions associées à la file :

```
int     initFile    (File_t** file, int taille);
int     fileVide    (File_t* file);
int     filePleine  (File_t* file);
int     enfiler     (File_t* file, T element);
int     defiler     (File_t* file, T* element);
void    libererFile (File_t* file);
void    debugFile   (File_t* file);
```

initFile : Allocation et initialisation de la tête de la file, allocation du tableau de la file.

fileVide : Teste si la file est vide ou non.

filePleine : Teste si la file est pleine ou non.

enfiler : Ajoute un élément à la fin de la file.

defiler : Retire et renvoie le premier élément de la file.

libererFile : Libère la mémoire du tableau et de la tête de la file.

debugFile : Affiche le contenu de la tête et du tableau de la file.

3) Main (truc.c)

Le programme principale contient 2 fonctions. La fonction TRUC donnée (en récursif) et la fonction TRUC_IT qui est la version itérative. Ce programme utilise donc le module de la pile.

```
void truc   (int i, int n, int* T);
void truc_it(int i, int n, int* T);
```

IV – Présentation du programme

1) Module Pile

cf. code de pile.h et pile.c en annexe.

2) Module File

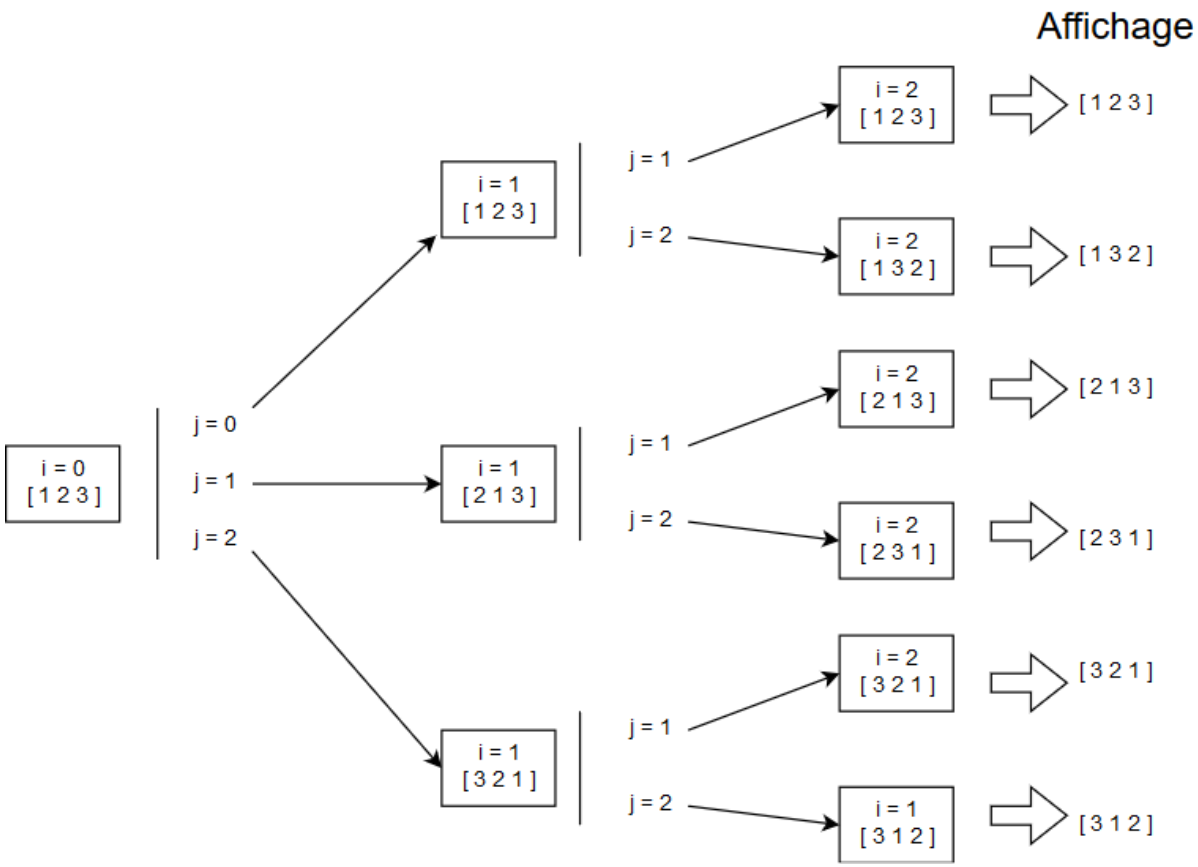
cf. code de file.h et file.c en annexe.

3) Programme de tests

Le programme du code du fichier « test.c » consiste à tester toutes les fonctionnalités et les limites des modules de pile et de file. C'est donc une seule fonction (main) qui créer une pile et une file et effectue des opérations dessus afin de détecter les différents comportements des modules pile et file. Pour plus de précision sur les tests effectués, cf. compte rendu d'exécution du programme de tests.

4) Programme de dérécursification

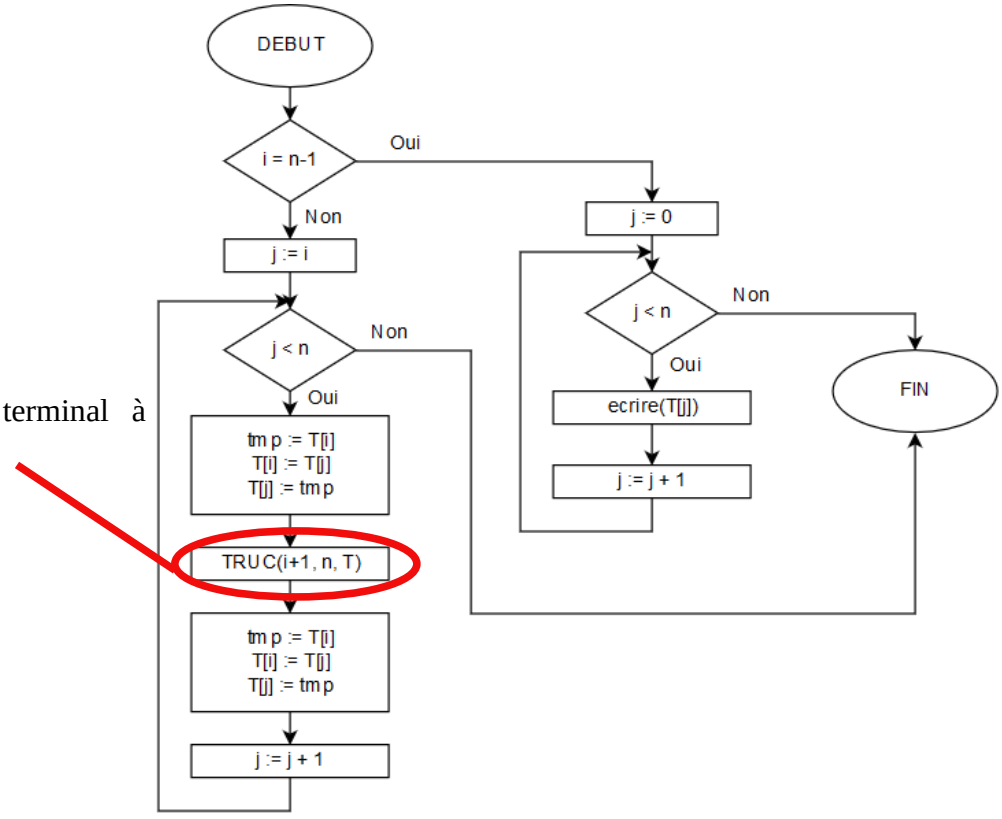
Le code du fichier « truc.c » consiste à implémenter en C la fonction TRUC de l'énoncé afin de comprendre son fonctionnement et dérécursifier cette fonction puis implémenter également la fonction itérative. Pour comprendre le fonctionnement de la fonction, une trace à d'abord été faite avec comme données d'entrées i = 0 ; n = 3 et T[3] = { 1, 2, 3 } :



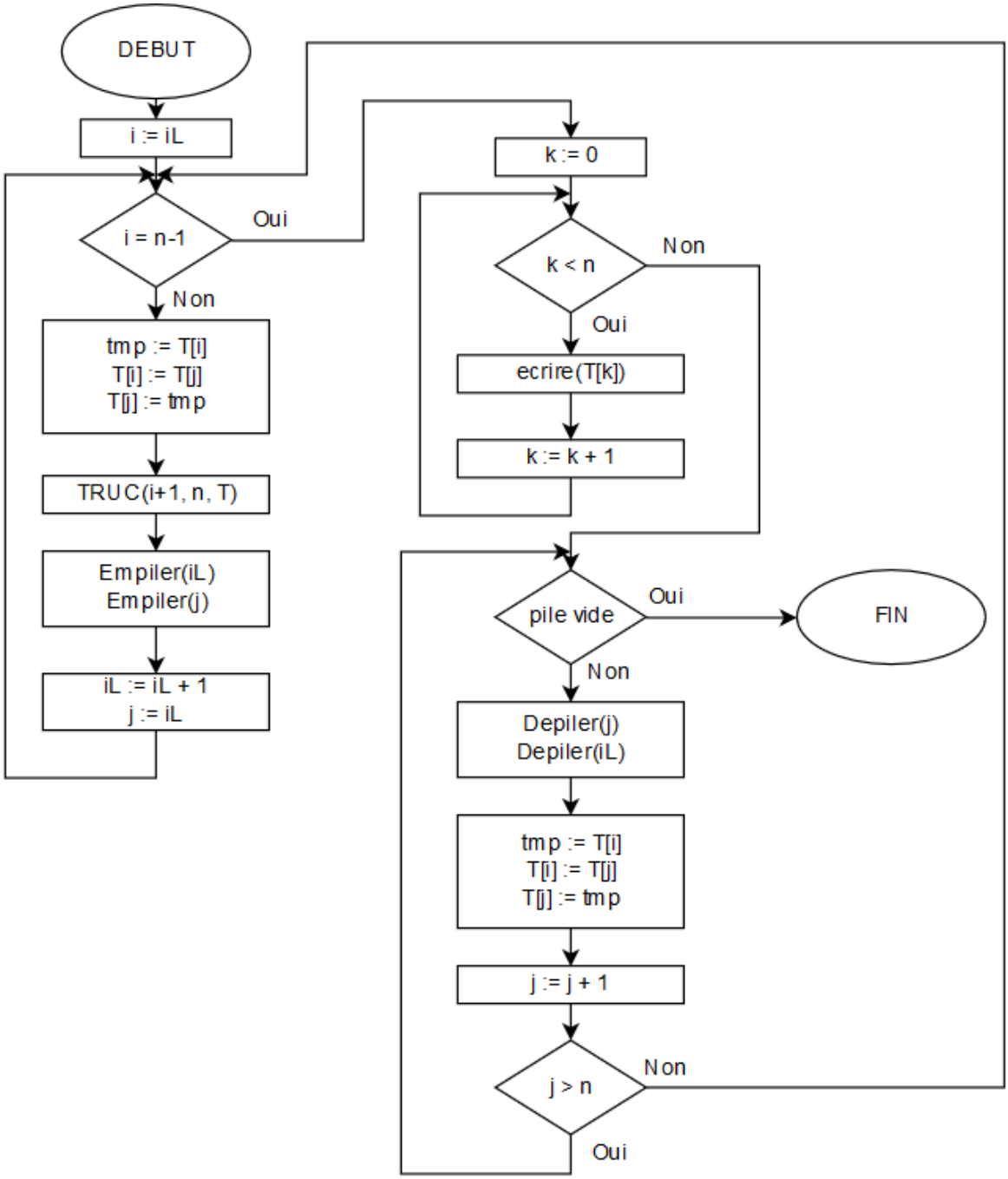
On déduit de cette trace que le but de cette fonction récursive est d'afficher toutes les permutations d'un tableau donné.

Organigramme correspondant à la fonction récursive :

Appel récursif non terminal à retirer



Grâce à la trace, on sait que le contexte à sauvegarder est la valeur de i et la valeur de j. On utilise donc une pile pour empiler ces contextes jusqu'à ce que i arrive en bout de tableau puis on affiche l'état actuel du tableau et on revient au contexte précédent et on ajoute 1 à j. Cela donne l'organigramme suivant :



V – Compte-rendu d'exécution

1) Tests sur la pile et la file

Liste des cas à traiter sur la pile et la file :

- dépiler (respectivement défiler) une pile (resp. une file) vide.
- empiler (resp. enfiler) des éléments normalement.
- empiler (resp. enfiler) des éléments dans une pile (resp. une file) pleine.
- dépiler (resp. défiler) des éléments normalement.
- Liberer la mémoire de la pile (resp. de la file)

Le programme « test.c » teste ces cas. Il créer une pile et une file, essaye de dépiler une pile vide, de défiler une file vide, de mettre trop d'éléments…

Test de dépiler, défiler une pile, file vide :

```
    if(!depiler(maPile, &var))
        printf("ERREUR DEPILER\n");
    if(!defiler(maFile, &var))
        printf("ERREUR DEFILER\n");
    debugPile(maPile);
    debugFile(maFile);
```

```
=== Tests dépiler, défiler sur une liste sans éléments ===
ERREUR DEPILER
ERREUR DEFILER
Pile:
        Taille max: 10
        Rang sommet: -1
        []
File:
        Taille max: 10
        Compteur: 0
        Rang premier: 0
        Rang dernier: 9
        []
```

Test d'empiler, enfiler des éléments normalement et empiler, enfiler des éléments dans une pile, file pleine :

```
    var = 89;
    for(i = 0; i < 15; i++)
    {
        if(!empiler(maPile, var))
            printf("ERREUR EMPILER\n");
        if(!enfiler(maFile, var))
            printf("ERREUR ENFILER\n");
    }
    debugPile(maPile);
    debugFile(maFile);
```

```
=== Tests empiler, enfiler trop d'éléments ===
ERREUR EMPILER
ERREUR ENFILER
ERREUR EMPILER
ERREUR ENFILER
ERREUR EMPILER
ERREUR ENFILER
ERREUR EMPILER
ERREUR ENFILER
ERREUR EMPILER
ERREUR ENFILER
Pile:
        Taille max: 10
        Rang sommet: 9
        [89, 89, 89, 89, 89, 89, 89, 89, 89, 89]
File:
        Taille max: 10
        Compteur: 10
        Rang premier: 0
        Rang dernier: 9
        [89, 89, 89, 89, 89, 89, 89, 89, 89, 89]
```

(la taille de la pile et de la file est de 10)

Pour également tester la plupart des situations, le programme rempli une pile, inverse son contenu à l'aide d'une file et affiche la pile et la file aux différentes étapes afin de vérifier si tous les comportements sont normaux.

```
============== Tests de validité des implémentations de la pile et de la file ==============
On teste la pile et la file en essayant d'inverser le contenu d'une pile à l'aide d'une file
-----------------------------------------------------------------------------------------------
On remplit la pile (avec les chiffres de 1 à 10 par exemple)
Pile:
        Taille max: 10
        Rang sommet: 9
        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
File:
        Taille max: 10
        Compteur: 0
        Rang premier: 0
        Rang dernier: 9
        []
-----------------------------------------------------------------------------------------------
On vide la pile et on enfile chaque élément qu'on dépile
Pile:
        Taille max: 10
        Rang sommet: -1
        []
File:
        Taille max: 10
        Compteur: 10
        Rang premier: 0
        Rang dernier: 9
        [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
-----------------------------------------------------------------------------------------------
On défile en remettant les éléments dans la pile
Pile:
        Taille max: 10
        Rang sommet: 9
        [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
File:
        Taille max: 10
        Compteur: 0
        Rang premier: 0
        Rang dernier: 9
        []
```

2) Fonction TRUC et TRUC_IT

Le programme test.c permet de vérifier que l'implémentation en code C de l'organigramme de la fonction truc dérécursifiée fonctionne et possède le même comportement que la fonction récursive.

```
int T[TAILLE]   = { 1, 2, 3 };

truc(0, TAILLE, T);
printf("---------\n");
truc_it(0, TAILLE, T);
```



```
[ 1 2 3 ]
[ 1 3 2 ]
[ 2 1 3 ]
[ 2 3 1 ]
[ 3 2 1 ]
[ 3 1 2 ]
---------
[ 1 2 3 ]
[ 1 3 2 ]
[ 2 1 3 ]
[ 2 3 1 ]
[ 3 2 1 ]
[ 3 1 2 ]
```

VI – Makefile

Afin de compiler les programmes, on utilise un makefile :

```
OPT = -g -Wall -Wextra
LOG = @echo "\#MAKE"
PF = pile-file
TRUC = truc

all: $(PF) $(TRUC)

$(TRUC): truc.c pile.o
    gcc -o $(TRUC) truc.c pile.o $(OPT)
    $(LOG) "executable truc généré"

$(PF): test.c pile.o file.o
    gcc -o $(PF) test.c pile.o file.o $(OPT)
    $(LOG) "excutable pile-file généré"

pile.o: pile.h pile.c
    gcc -o pile.o -c pile.c $(OPT)
    $(LOG) "lib pile généré"

file.o: file.h file.c
    gcc -o file.o -c file.c $(OPT)
    $(LOG) "lib file généré"

clean:
    rm *.o
    rm $(PF)
    rm $(TRUC)
```

Ce makefile génère 2 exécutables : pile-file et truc.

pile-file est le programme de tests pour la pile et la file et truc est le programme testant la fonction truc et sa version itérative.

L'exécutable pile-file a donc besoin du module de pile et de file (donc pile.o et file.o) et du code source de test (test.c).

L'exécutable truc a besoin du module de pile (pile.o) et du code des fonctions truc et du main pour les tester (truc.c).

Commandes :

- "make" pour faire les 2 exécutables
- "make truc" pour juste créer l'exécutable truc
- "make pile-file" pour juste créer l'exécutable pile-file
- "make clean" pour effacer les exécutables et les .o
- "./truc" ou "./pile-file" pour lancer le programme