SDD TP2 : Pile et File

Cavani Nicolas & Leduque Adrien

Mars 2020

Table des matières

1 Introduction

L'objectif de ce TP est de concevoir une bibliothèque de fonction pour la manipulation des structures Pile et File. Ces structures doivent être de type modulable.

2 Utilisation

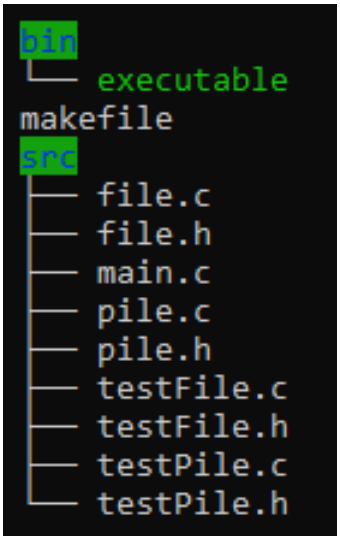
Le type de la pile et de la file est modulable. Il vous suffit de mettre le type souhaiter dans le typedef qui se situe au début du fichier pile.h et file.h. Pour afficher la pile (resp. la file), vous devez créer une fonction qui affiche un élément du type voulu que vous passerez en paramètre de la fonction afficherPile (resp. afficherFile).

3 Arborescence du projet

— main.c : Contient le programme principal d'inversion d'une pile ainsi que l'appel des fonctions de test unitaire.

— pile.h : Déclaration des fonctions de manipulation de la pile.
— pile.c : Définition des fonctions de manipulation de la pile.
— file.h : Déclaration des fonctions de manipulation de la file.
— file.c : Définition des fonctions de manipulation de la file.

— testPile.h : Déclaration des tests des fonctions pile.
— testPile.c : Définition des tests des fonctions pile.
— testFile.h : Déclaration des tests des fonctions file.
— testFile.c : Définition des tests des fonctions file.



4 Makefile

Un Makefile est un fichier constitué de plusieurs règles de la forme :
cible : dépendances
    commandes

4.1 Cibles :

all : regroupe dans ses dépendances l'executable à produire.
clean : permet de supprimer tous les fichiers intermédiaires.
.PHONY : permet de reconstruire les dépendances de la cible (dépendante de .PHONY), dans le cas où des fichiers porteraient le même nom.

4.2 Variables :

CFLAGS : regroupe les options de compilation.
    -Wall -Wextra : warnings de compilation
    -g : Génère les informations de débogage.
    -MMD : permet la génération des fichiers .d pour les dépendances.
LIB : regroupe les options de l'édition de liens.
    -lm : bibliothèque maths
SRC : contient la liste des fichiers sources du projet. La commande wildcard permet l'utilisation du joker * dans la définition d'une variable.
OBJ : contient la liste des fichiers objets. OBJ est rempli à partir de SRC.
    $(patsubst pattern, replacement, text) : Trouve les mots dans text qui correspondent au pattern et les remplace par replacement.
DEP : fichiers .d contenant les dépendances associées à un fichier .o du même nom.

4.3 Variables internes :

$@ : Le nom de la cible.
$< : Le nom de la première dépendance.
$^ : La liste des dépendances.
@ : Permet de ne pas afficher la commande dans la console.

5 Pile

La Pile est une structure qui contient plusieurs champs :
capacite (int) : Capacité de la pile.
sommet (int) :
base (type *) : adresse du début de la file.

5.1 initPile()

Initialise une pile de taille le paramètre taille.

5.2 estVidePile()

Indique si la pile passée en paramètre est vide ou non.

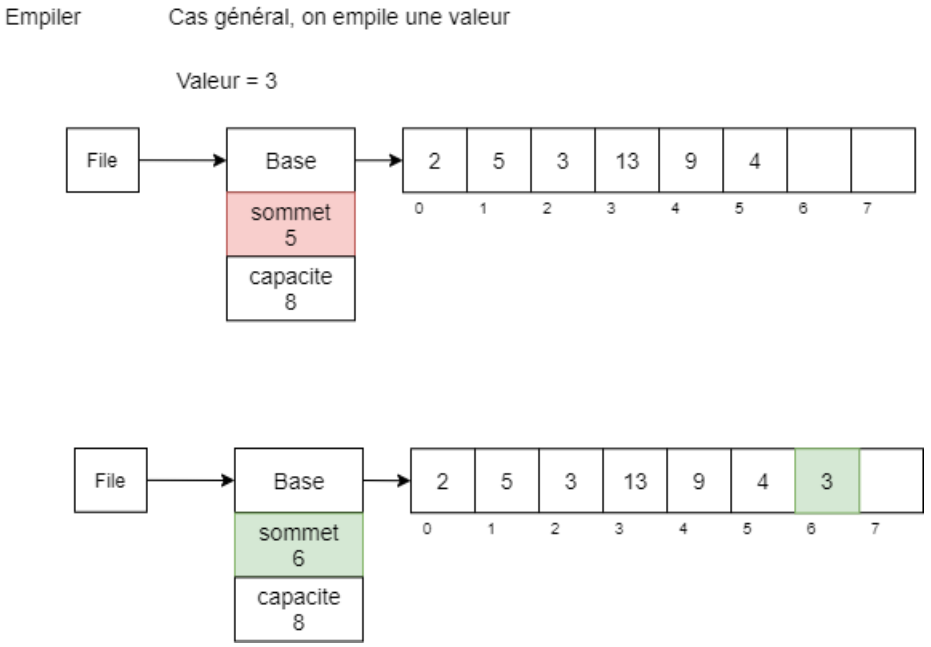
5.3 estPleinePile()

Indique si la pile passée en paramètre est pleine ou non.

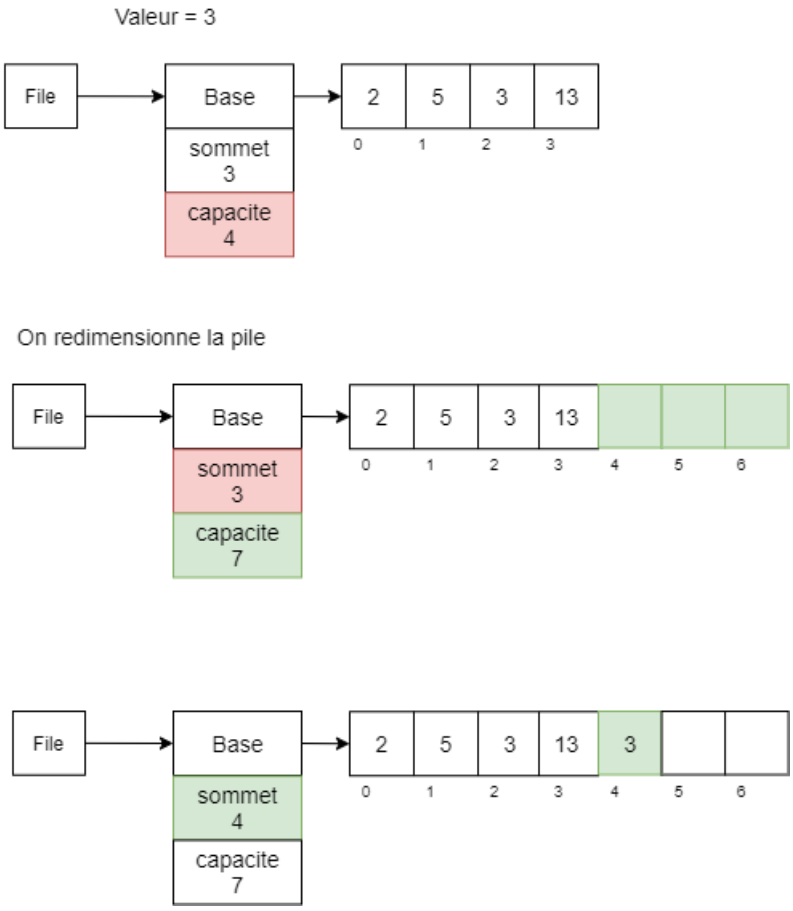
5.4 empiler()

Empile une valeur à la fin de la pile, et l'agrandit si elle est trop petite.

5.4.1 Traces



Empiler On empile une valeur dans une pile pleine, la pile est redimensionnée

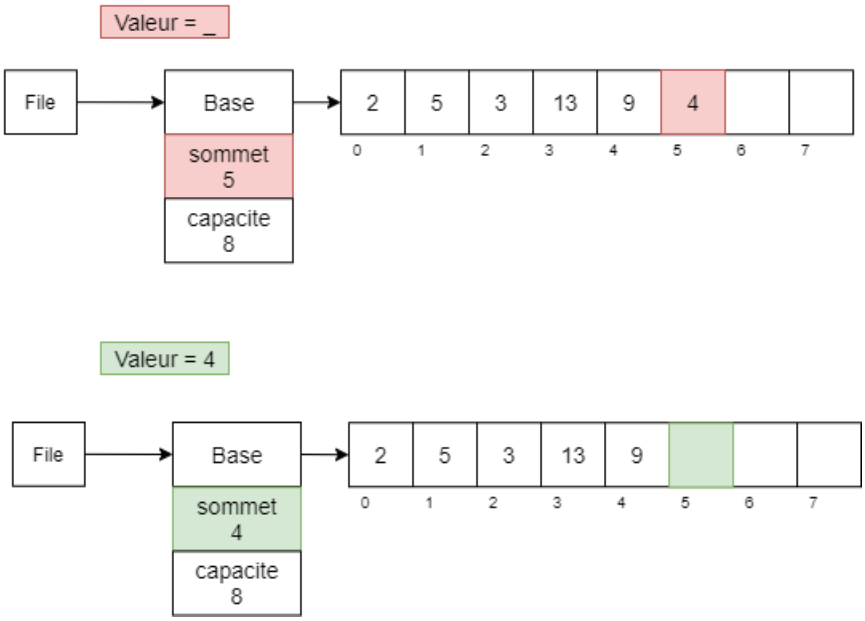


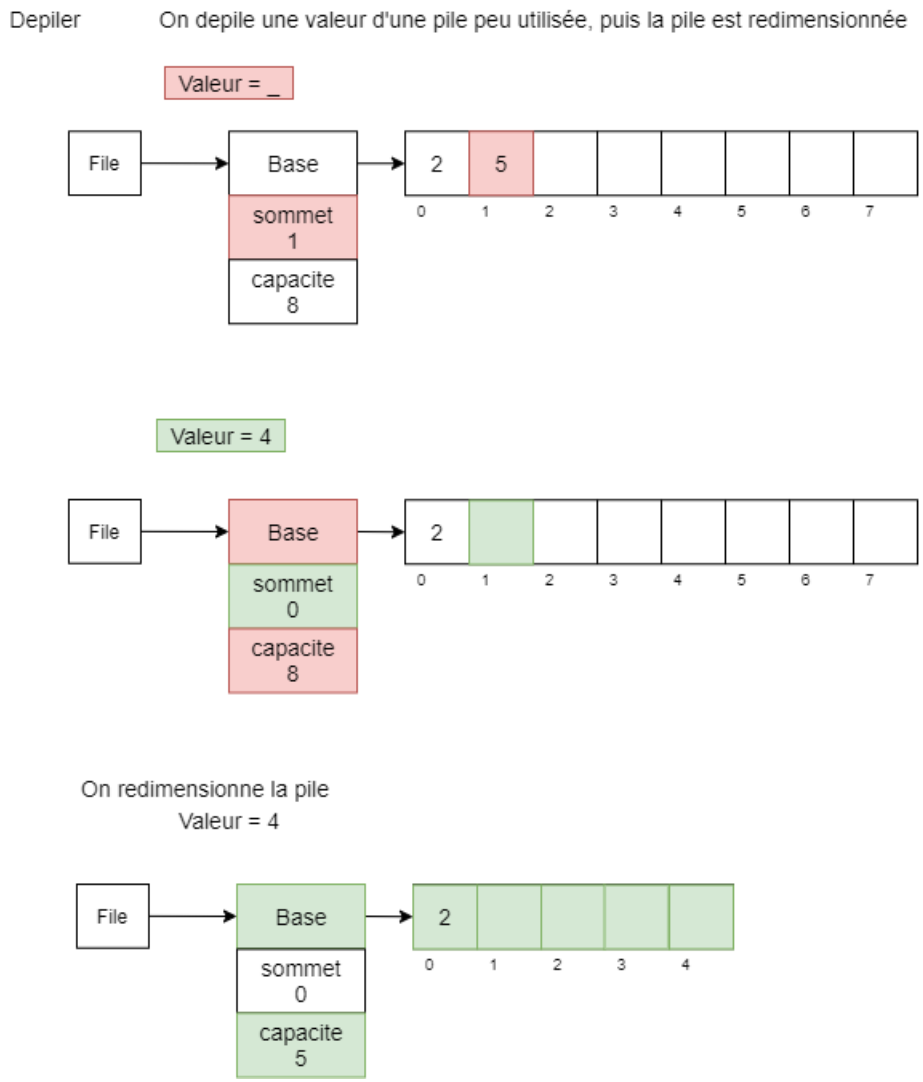
5.5 depiler()

Défile la valeur en tête de la file, et la réduit si elle peu utilisée.

5.5.1 Traces

Depiler Cas général, on depile une valeur





5.6 redimensionnerPile()

Redimensionne la pile en copiant les valeurs.

5.7 libererPile()

Libère la mémoire utilisée par la pile.

5.8 afficherPile()

Affiche la pile.

5.9 affichePileInt()

Affiche un int.

5.10 afficherPileChar()

Affiche un char.

5.11 afficherPileChaineChar()

Affiche une chaîne de caractères.

6 File

La file est une structure qui contient plusieurs champs :
capacite : Capacité maximal d'élément que peut contenir la pile.
nbElements (int) : Nombre d'élément que contient la file.
indexInsertion (int) :
indexSuppression (int) :
base (type*) : adresse du début de la file.

6.1 initFile()

Initialise une file de taille le paramètre taille.

6.2 estVideFile()

Indique si la file passée en paramètre est vide ou non.

6.3 estPleineFile()

Indique si la file passée en paramètre est pleine ou non.

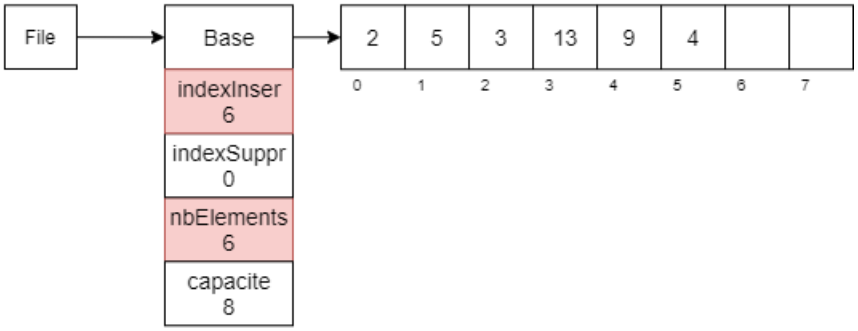
6.4 enfiler()

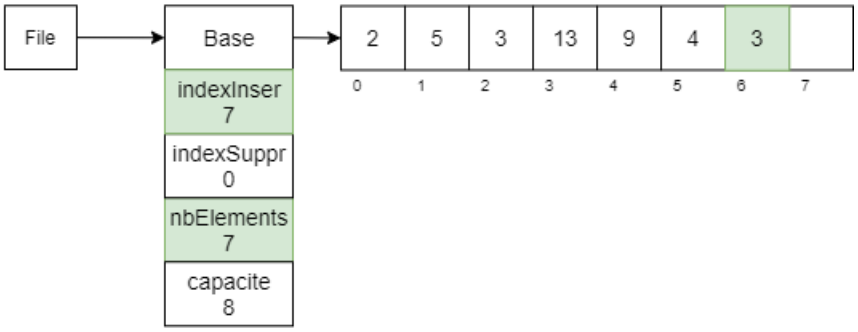
Enfile une valeur à la fin de la file, et l'agrandit si elle est trop petite.

6.4.1 Traces

Enfiler Cas général, on enfile une valeur

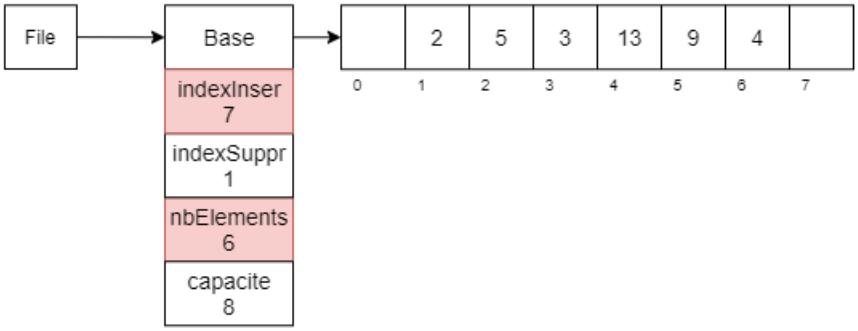
Valeur = 3

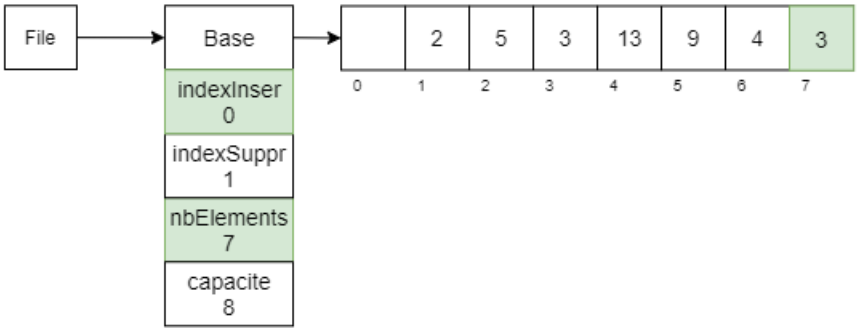


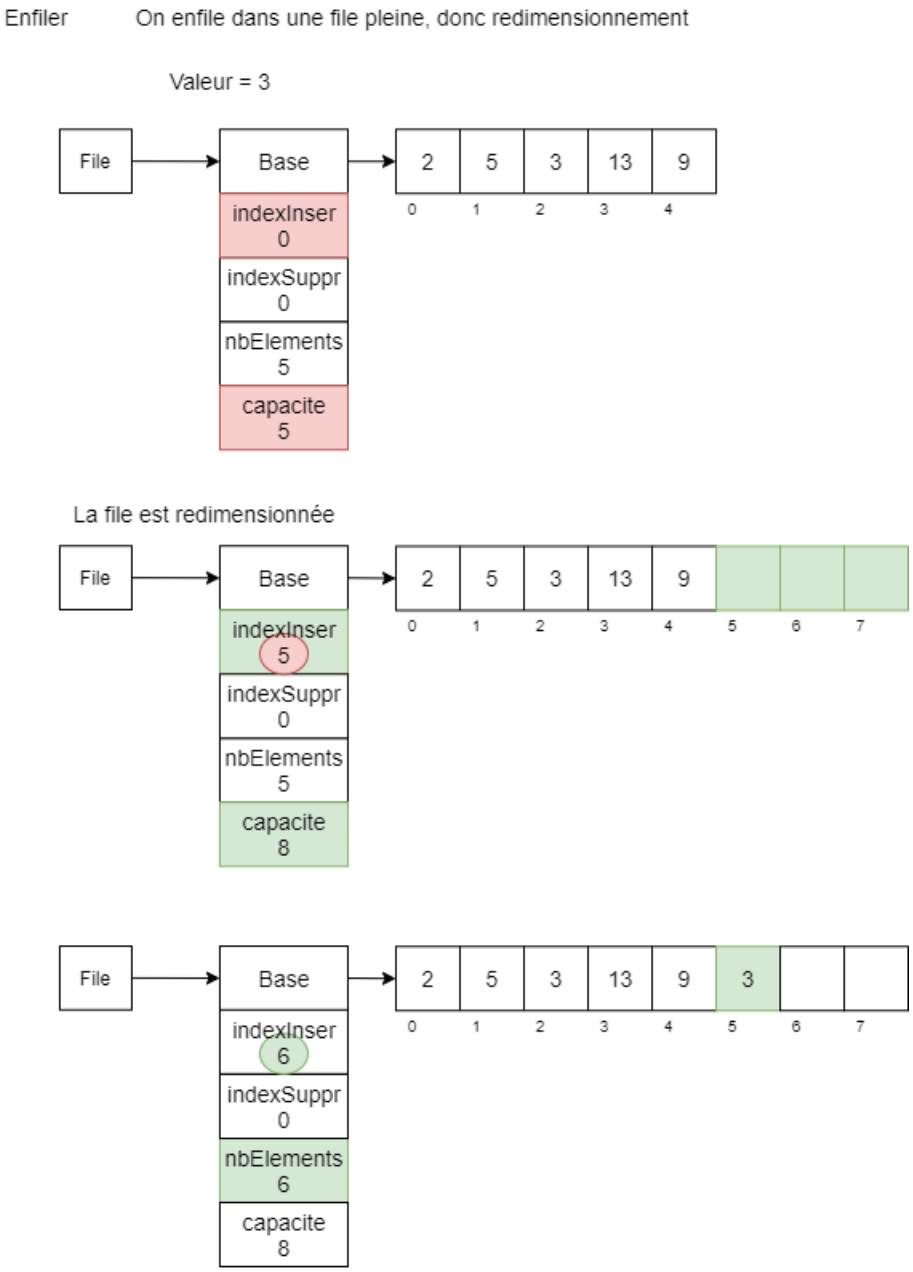


Enfiler On enfile une valeur, l'indexInsertion repasse au début de la liste contigue

Valeur = 3



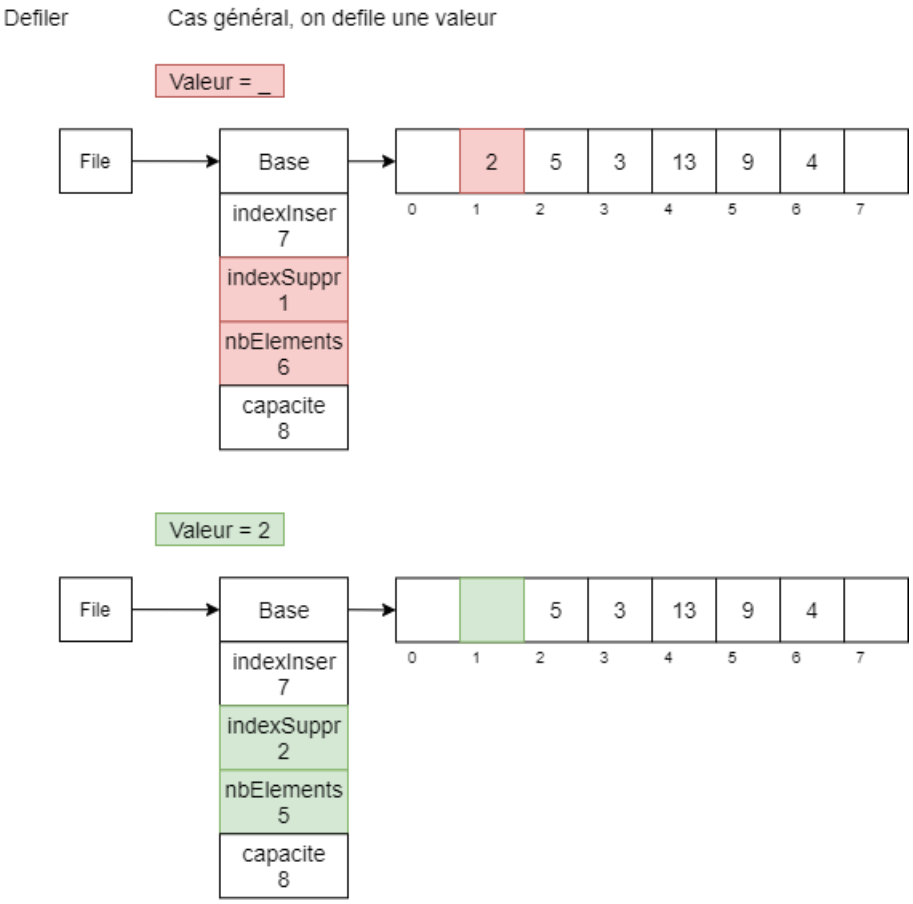


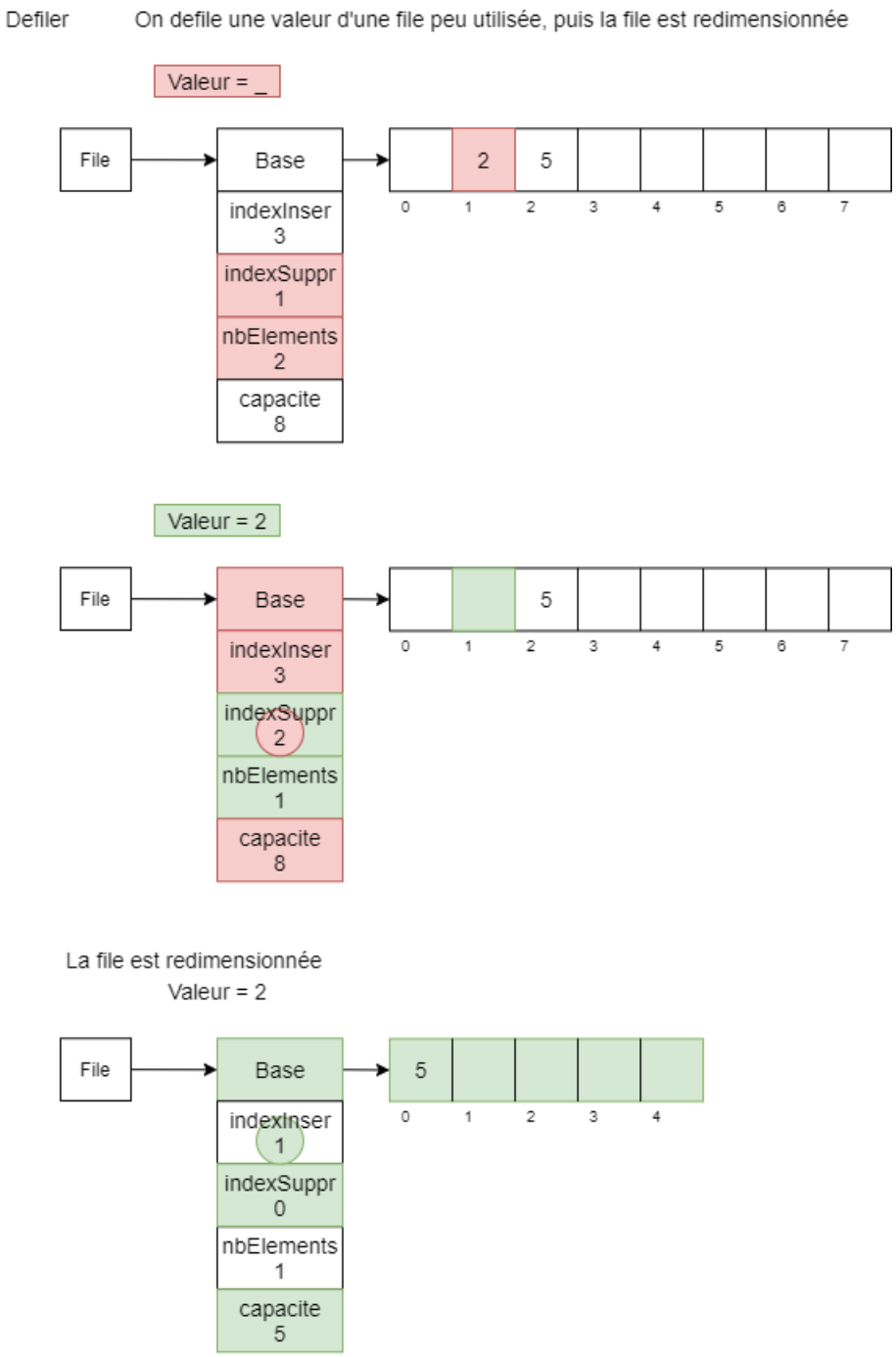


6.5 defiler()

Défile la valeur en tête de la file, et la réduit si elle peu utilisée.

6.5.1 Traces


Defiler
Cas général, on defile une valeur
Valeur = _
File
Base
indexInser
7
indexSuppr
1
nbElements
6
capacite
8
2
5
3
13
9
4
0
1
2
3
4
5
6
7
Valeur = 2
File
Base
indexInser
7
indexSuppr
2
nbElements
5
capacite
8
5
3
13
9
4
0
1
2
3
4
5
6
7



6.6 redimensionnerFile()

Redimensionne la file en copiant les valeurs.

6.7 libererFile()

Libère la mémoire utilisée par la file.

6.8 afficherFile()

Affiche la file.

6.9 afficherFileInt()

Affiche un int.

6.10 afficherFileChar()

Affiche un char.

6.11 afficherFileChaineChar()

Affiche une chaîne de caractères.

7 Main

Le programme à deux modes d'exécution :
— Inversion d'une pile écrite dans le code.
Exécuter avec ./bin/excutable
— Test du fonctionnement des fonctions de manipulation des piles et des files.
Exécuter avec ./bin/executable 1

8 TestPile

Les fonctions présentes dans ce fichier contiennent tous les tests pour vérifier que les fonctions de manipulation des piles fonctionnes. Chaque fonction de ce fichier test une unique manipulation.

8.1 testUnitairePile()

Fonction appelant toutes les fonctions de test de la pile.

8.2 testInitPile()

Fonction qui test la fonction d'initialisation de la pile.

8.3 testEstVidePile()

Fonction qui test la fonction testant si la pile est vide.

8.4 testEstPleinePile()

Fonction qui test la fonction testant si la pie est pleine.

8.5 testEmpiler()

Fonction qui test la fonction d'empilage de la pile.

8.6 testDepiler()

Fonction qui test la fonction de dépilage de la pile.

9 TestFile

Les fonctions présentes dans ce fichier contiennent tous les tests pour vérifier que les fonctions de manipulation des files fonctionnes. Chaque fonction de ce fichier test une unique manipulation.

9.1 testUnitaireFile()

Fonction appelant toutes les fonctions de test de la file.

9.2 testInitFile()

Fonction qui test la fonction d'initialisation de la file.

9.3 testFileEstVide()

Fonction qui test la fonction testant si la file est vide.

9.4 testFileEstPleine()

Fonction qui test la fonction testant si la file est pleine.

9.5 testEnfiler()

Fonction qui test la fonction d'enfilage de la file.

9.6 testDefiler()

Fonction qui test la fonction de défilage de la file.

10 Cas d'exécution

10.1 Pile

10.1.1 Initialisation pile

— Initialisation de la pile

10.1.2 Pile est vide

— La pile vient d'être crée
— La pile contient un élément
— La pile est vide après avoir été non vide
— La pile est pleine

10.1.3 Pile est pleine

— La pile vient d'être crée
— La pile contient un élément
— La pile est pleine
— La pile n'est plus pleine après l'avoir été
— La pile est de nouveau pleine après ne plus l'avoir été

10.1.4 Empiler

— La pile vient d'être crée
— La pile est partiellement pleine
— La pile est pleine, redimensionnement
— La pile a une capacité de 0

10.1.5 Dépiler

— La pile vient d'être crée
— La pile est partiellement pleine
— La pile est pleine
— La pile contient 1 élément et on doit redimensionné
— La pile est partiellement pleine et on doit redimensionné
— La pile a une capacité de 0

10.2 File

10.2.1 Initialisation file

— Initialisation de la file

10.2.2 File est vide

— La file vient d'être crée
— La file est partiellement pleine
— La file est vide après avoir été non vide

10.2.3 File est pleine

— La file vient d'être crée
— La file est partiellement pleine
— La file est pleine
— La file n'est plus pleine après l'avoir été
— La file est pleine après ne plus l'avoir été

10.2.4 Enfiler

— La file vient d'être crée
— La file est partiellement pleine sans avoir été défilé
— La file est partiellement pleine en ayant été défilé
— La file est pleine sans avoir été défilé, redimensionnement
— La file est pleine en ayant été défilé , redimensionnement

10.2.5 Défiler

— La file vient d'être crée
— La file est vide en ayant été défilé
— La file est partiellement pleine sans avoir été défilé
— La file est partiellement pleine en ayant été défilé
— La file contient un élément et doit être redimensionné
— La file est partiellement pleine et doit être redimensionné

11 Annexe

11.1 Pile

11.1.1 initPile

```
25  /* ------------------------------------------------------------ */
26  /* initPile     Initialise une pile de taille le paramètre taille */
27  /*                                                              */
28  /* En entrée :  taille (int) : Taille de la pile                */
29  /*                                                              */
30  /* En sortie :  pile : pointeur sur la pile (pile_t *)          */
31  /* ------------------------------------------------------------ */
32  pile_t * initPile(int);
```

```
11  pile_t * initPile(int capacite) {
12
13      /* Allocation de la pile */
14      pile_t * pile = (pile_t*) malloc(sizeof(pile_t));
15
16      if (pile != NULL) {
17          (*pile).base = NULL;
18
19          /* Allocation de la base de la pile */
20          (*pile).base = (type*)malloc(capacite*sizeof(type));
21
22          if ((*pile).base != NULL) {
23              (*pile).capacite = capacite;
24              (*pile).sommet = -1;
25
26          } else {    /* Si l'allocation c'est pas faite, on libère la pile */
27              free(pile);
28              pile = NULL;
29          }
30      }
31      return pile;
32  }
```

11.1.2 estVidePile & estPleinePile

```
34  /* ------------------------------------------------------------------ */
35  /* EstVidePile  Indique si la pile passée en paramètre est vide ou non */
36  /*                                                                    */
37  /* En entrée :  pile (pile_t *) : Pile à tester                       */
38  /*                                                                    */
39  /* En sortie :  (int) booléen : 1 si la pile est vide, 0 sinon        */
40  /* ------------------------------------------------------------------ */
41  int estVidePile(pile_t *);
```

```
43  /* ------------------------------------------------------------------ */
44  /* estPleinePile  Indique si la pile passée en paramètre est pleine ou non */
45  /*                                                                    */
46  /* En entrée :  pile (pile_t *) : Pile à tester                       */
47  /*                                                                    */
48  /* En sortie :  (int) booléen : 1 si la pile est pleine, 0 sinon      */
49  /* ------------------------------------------------------------------ */
50  int estPleinePile(pile_t *);
```

```c
int estVidePile(pile_t * pile) {
    return (*pile).sommet == -1;
}


int estPleinePile(pile_t * pile) {
    return (*pile).capacite == (*pile).sommet + 1;
}
```

11.1.3 empiler

```c
/* ------------------------------------------------------------------ */
/* empiler      Empile une valeur au sommet de la pile,               */
/*                          et l'aggrandit si elle est trop petite    */
/*                                                                    */
/* En entrée :  pile (pile_t *) : Pile à remplir                      */
/*              valeur (type)   : Elément à empiler                   */
/*                                                                    */
/* En sortie :  codeErreur (int) booléen : 1 si erreur, 0 sinon       */
/* ------------------------------------------------------------------ */
int empiler(pile_t *, type);
```

```c
int empiler(pile_t * pile, type v) {
    int codeErreur = 1;

    /* Si la pile est pleine, on redimensionne, sinon, on empile */
    if (estPleinePile(pile)) {

        /* + 1 pour le cas ou la capacite vaut 1 */
        int nvCapacite = 1.5*(*pile).capacite + 1;

        /* Si le redimensionnement a fonctionne, on empile la valeur */
        if (!redimensionerPile(pile, nvCapacite))
            codeErreur = empiler(pile, v);

    } else {
        codeErreur = 0;
        (*pile).sommet ++;
        (*pile).base [(*pile).sommet] = v;
    }

    return codeErreur;
}
```

11.1.4 depiler

```c
/* ------------------------------------------------------------------ */
/* depiler      Dépile la valeur au sommet de la pile,                */
/*                          et la réduit si elle peu utilisée         */
/*                                                                    */
/* En entrée :  pile (pile_t *) : Pile à vider                        */
/*              valeur (type)   : Elément dépilé                      */
/*                                                                    */
/* En sortie :  codeErreur (int) booléen : 2 si erreur, 0 sinon       */
/* ------------------------------------------------------------------ */
int depiler(pile_t *, type *);
```

```c
int depiler(pile_t * pile, type * v) {
    int codeErreur = 1;

    /* Si la pile est non vide, on depile */
    if (!estVidePile(pile)) {
        codeErreur = 0;
        *v = (*pile).base [(*pile).sommet];
        (*pile).sommet --;

        /* Si la pile est tres peu utilise, on diminue la taille */
        if ((*pile).sommet < (int)0.25*(*pile).capacite) {
            int nvCapacite = 0.5*(*pile).capacite + 1;
            redimensionerPile(pile, nvCapacite);
        }
    }

    return codeErreur;
}
```

11.1.5 redimensionnerPile

```c
/* ------------------------------------------------------------ */
/* redimensionerPile    Redimensionne la base de la pile        */
/*                                                              */
/* En entrée : pile (pile_t *) : Pile à redimensionner          */
/*             nvCapacité (int) : Taille de la nouvelle base    */
/*                                                              */
/* En sortie : codeErreur (int) booléen : 1 si erreur, 0 sinon  */
/* ------------------------------------------------------------ */
int redimensionerPile(pile_t *, int);
```

```c
int redimensionerPile(pile_t * pile, int nvCapacite) {
    int codeErreur = 1;

    /* On realloc la base */
    type * nvBase = realloc((*pile).base, nvCapacite*sizeof(type));
    if (nvBase != NULL) {
        codeErreur = 0;
        /* Si l'allocation c'est bien faite, on modifie la base et la capacite */
        (*pile).base = nvBase;
        (*pile).capacite = nvCapacite;
    }
    return codeErreur;
}
```

11.1.6 afficherPile

```c
/* ------------------------------------------------------------*/
/* afficherPile     Affiche la pile                            */
/*                                                             */
/* En entrée : pile (pile_t *) : Pile à afficher               */
/*             pfAfficher (void (*)(type)) : pointeur sur      */
/*                      la fonction d'affichage du bon type    */
/* En sortie : void                                            */
/* ------------------------------------------------------------*/
void afficherPile(pile_t *, void (*) (type));
```

```c
/* ------------------------------------------------- */
/* afficherPileInt      Affiche un élément de type int */
/*                                                   */
/* En entrée : nombre (int) : Nombre à afficher      */
/*                                                   */
/* En sortie : void                                  */
/* ------------------------------------------------- */
void afficherPileInt(int);

/* ------------------------------------------------- */
/* afficherPileChar   Affiche un élément de type char */
/*                                                   */
/* En entrée : caractere (char) : Caractère à afficher */
/*                                                   */
/* En sortie : void                                  */
/* ------------------------------------------------- */
void afficherPileChar(char);

/* ------------------------------------------------------------------------ */
/* afficherPileChaineCarac      Affiche un élément de type chaine de caractère */
/*                                                                          */
/* En entrée : chaine (char *) : Chaine de caractères à afficher            */
/*                                                                          */
/* En sortie : void                                                         */
/* ------------------------------------------------------------------------ */
void afficherPileChaineCarac(char *);
```

```c
void afficherPile(pile_t * pile, void (*pfAfficher) (type)) {

    printf("Pile : capacite=%d\n", (*pile).capacite);
    printf("       ");

    if(!estVidePile(pile)) {
        /* On affiche chaque element un par un en appellant la fonction d'afichage */
        for (int i=0; i <=(*pile).sommet; i++)
            (*pfAfficher) ( (*pile).base[i] );
    } else {
        printf("vide");
    }

    printf("\n");
}


void afficherPileInt(int nombre) {
    printf("%d ", nombre);
}


void afficherPileChar(char caractere) {
    printf("%c ", caractere);
}


void afficherPileChaineCarac(char * chaine) {
    printf("%s ", chaine);
}
```

11.2 File

11.2.1 initFile

```
/* ------------------------------------------------------------------- */
/* initFile     Initialise une file de taille le paramètre taille      */
/*                                                                     */
/* En entrée :  file : file_t * : File à tester                        */
/*                                                                     */
/* En sortie :  file : Pointeur sur la file (file_t *)                 */
/* ------------------------------------------------------------------- */
file_t * initFile(int);
```

```
file_t * initFile(int capacite) {
    file_t * file = (file_t *)malloc(sizeof(file_t)); /*File créée*/

    if (file != NULL) {
        file->base = (type *)malloc(capacite * sizeof(capacite));

        /* Erreur de memoire */
        if (file->base == NULL) {
            free(file);
            file = NULL;

        /* Initialisation de la file */
        } else {
            file->capacite         = capacite;
            file->nbElements       = 0;
            file->indexSuppression = 0;
            file->indexInsertion   = 0;
        }
    }

    return file;
}
```

11.2.2 estVideFile & estPleineFile

```
/* ------------------------------------------------------------------- */
/* estVideFile  Indique si la file passée en paramètre est vide ou non */
/*                                                                     */
/* En entrée :  file (file_t *) : File à tester                        */
/*                                                                     */
/* En sortie :  (char) booléen : 1 si la file est vide, 0 sinon        */
/* ------------------------------------------------------------------- */
char estVideFile(file_t *);
```

```
/* ------------------------------------------------------------------- */
/* estPleineFile  Indique si la file passée en paramètre est pleine ou non */
/*                                                                     */
/* En entrée :  file (file_t *) : File à tester                        */
/*                                                                     */
/* En sortie :  (char) booléen : 1 si la file est pleine, 0 sinon      */
/* ------------------------------------------------------------------- */
char estPleineFile(file_t * file);
```

```c
char estVideFile(file_t * file) {
    return (file->nbElements == 0);
}


char estPleineFile(file_t * file) {
    return (file->nbElements == file->capacite);
}
```

11.2.3 enfiler

```c
/* ------------------------------------------------------------------ */
/* enfiler      Enfile une valeur à la fin de la file,                */
/*                          et l'aggrandit si elle est trop petite    */
/*                                                                    */
/* En entrée :  file (file_t *) : File à remplir                      */
/*              valeur (int)    : Valeur à enfiler                    */
/*                                                                    */
/* En sortie :  codeErreur (char) booléen : 1 si erreur, 0 sinon      */
/* ------------------------------------------------------------------ */
char enfiler(file_t * file, type valeur);
```

```c
char enfiler(file_t * file, type valeur) {
    char codeErreur = 1; /*Code erreur*/

    /* Si la file n'est pas déjà remplie, on enfile la valeur */
    if (!estPleineFile(file)) {
        file->base[file->indexInsertion] = valeur;
        file->indexInsertion = (file->indexInsertion + 1) % file->capacite; /*Index d'insertion suivante*/

        file->nbElements += 1;
        codeErreur = 0;

    /* Sinon, il faut redimensionner la file */
    } else {
        int nouvCapacite = 1.5 * file->capacite + 1;

        /* Si la redimension est sans erreur, on enfile la valeur */
        if (redimensionnerFile(file, nouvCapacite)) {
            printf("Redimensionnement impossible\n");
            codeErreur = 1;

        } else {
            enfiler(file, valeur);
            codeErreur = 0;
        }

    }
    return codeErreur;
}
```

11.2.4 defiler

```
/* ------------------------------------------------------------ */
/* defiler      Défile la valeur en tête de la file,            */
/*                          et la réduit si elle peu utilisée   */
/*                                                              */
/* En entrée :  file (file_t *) : File à vider                  */
/*              valeur (int)    : Valeur dépifée (par adresse)  */
/*                                                              */
/* En sortie :  codeErreur (char) booléen : 2 si erreur, 0 sinon */
/* ------------------------------------------------------------ */
char defiler(file_t * file, type * valeur);
```

```
char defiler(file_t * file, type * valeur) {
    char codeErreur = 2; /*Code erreur*/

    /* Si la file n'est pas vide, on defile la valeur */
    if (!estVideFile(file)) {
        *valeur = file->base[file->indexSuppression];
        file->indexSuppression = (file->indexSuppression + 1) % file->capacite;
        file->nbElements -= 1;
        codeErreur = 0;

        /* Si la file est peu utilisée, on la redimensionne */
        if (file->nbElements < 0.25 * file->capacite) {
            int nouvCapacite = 0.5 * file->capacite + 1;

            if (redimensionnerFile(file, nouvCapacite)) {
                printf("Redimensionnement impossible\n");
                codeErreur = 2;
            }
        }

    } else {
        printf("File vide\n");
    }

    return codeErreur;
}
```

11.2.5 redimensionnerFile

```
/* ------------------------------------------------------------------ */
/* redimensionnerFile   Redimensionne la file en copiant les valeurs  */
/*                                                                    */
/* En entrée :  file (file_t *)    : File à redimmensionner           */
/*              nouvCapacite (int) : Taille de la nouvelle file       */
/*                                                                    */
/* En sortie :  codeErreur (char) booléen : 1 si erreur, 0 sinon      */
/* ------------------------------------------------------------------ */
char redimensionnerFile(file_t * file, int nouvCapacite);
```

```c
char redimensionnerFile(file_t * file, int nouvCapacite) {
    char codeErreur = 1; /*Code Erreur*/
    int  i          = 0; /*Compteur*/

    type * nouvBase = (type *)malloc(nouvCapacite * sizeof(type));

    /* Si l'allocation est sans erreur, on copie la file */
    if (nouvBase != NULL) {
        /* On copie l'ancienne file dans la nouvelle (redimensionnée) */
        for (i=0; i<file->nbElements; i++) {
            nouvBase[i] = file->base[(file->indexSuppression + i) % file->capacite];
        }

        /* On libère l'ancienne file et on actualise les données de la nouvelle */
        free(file->base);
        file->base = nouvBase;

        /* On actualise les données de la file */
        file->indexSuppression = 0;
        file->indexInsertion = file->nbElements;
        file->capacite = nouvCapacite;

        codeErreur = 0;
    }
    return codeErreur;
}
```

11.2.6 afficherPile

```c
/* ------------------------------------------------------------ */
/* afficherFile   Affiche la file passée en paramètre           */
/*                                                              */
/* En entrée :  file (file_t *) : File à afficher               */
/*              pfAfficher (void (*)(type)) : pointeur sur      */
/*                       la fonction d'affichage du bon type    */
/*                                                              */
/* En sortie :  void                                            */
/* ------------------------------------------------------------ */
void afficherFile(file_t * file, void (*pfAfficher) (type));
```

```c
/* ------------------------------------------------------- */
/* afficherFileInt   Affiche un entier                     */
/*                                                         */
/* En entrée :  nombre (int) : int à afficher              */
/*                                                         */
/* En sortie :  void                                       */
/* ------------------------------------------------------- */
void afficherFileInt(int nombre);


/* ------------------------------------------------------- */
/* afficherFileChar   Affiche un caractère                 */
/*                                                         */
/* En entrée :  file (char) : char à afficher              */
/*                                                         */
/* En sortie :  void                                       */
/* ------------------------------------------------------- */
void afficherFileChar(char caractere);


/* ------------------------------------------------------- */
/* afficherFileChaineChar   Affiche une chaine de caractères */
/*                                                         */
/* En entrée :  chaine (char *) : Chaine à afficher        */
/*                                                         */
/* En sortie :  void                                       */
/* ------------------------------------------------------- */
void afficherFileChaineChar(char * chaine);
```

```c
void afficherFile(file_t * file, void (*pfAfficher) (type)) {
    int i = 0; /*Compteur*/

    if (!estVideFile(file)) {
        for (i=0; i<file->nbElements; i++) {
            pfAfficher(file->base[(file->indexSuppression + i) % file->capacite]);
        }

    } else {
        printf("File vide\n");
    }
}


void afficherFileInt(int nombre) {
    printf("%d\n", nombre);
}


void afficherFileChar(char caractere) {
    printf("%c\n", caractere);
}


void afficherFileChaineChar(char * chaine) {
    printf("%s\n", chaine);
}
```

11.3 Main

```c
int main(int argc, char const *argv[]) {

    if (argc >= 2 && *argv[1] == '1') {

        if (testUnitairePile() == 1)
            printf("\033[33m     Fonctions pile \033[32mfonctionne\033[00m\n\n");
        else
            printf("\033[33m     Fonctions pile \033[31mne fonctionne pas\033[00m\n\n");


        if (testUnitaireFile())
            printf("\033[33m     Fonctions file \033[32mfonctionne\033[00m\n\n");
        else
            printf("\033[33m     Fonctions file \033[31mne fonctoinne pas\033[00m\n\n");

    } else {

        int n = 26;
        pile_t * pile = initPile(10);

        for (int i=0; i<n; i++)
            empiler(pile, 97+i);

        afficherPile(pile, afficherPileChar);
        if (!inverserPile(pile))
            afficherPile(pile, afficherPileChar);
        else
            printf("Erreur lors de l'inversion de la pile\n");

        libererPile(pile);
    }

    return 0;
}
```