

SDD TP2 : Pile et File

Cavani Nicolas & Leduque Adrien

Mars 2020



Table des matières

1	Introduction	4
2	Utilisation	4
3	Arborescence du projet	4
4	Makefile	5
4.1	Cibles :	5
4.2	Variables :	5
4.3	Variables internes :	5
5	Pile	6
5.1	initPile()	6
5.2	estVidePile()	6
5.3	estPleinePile()	6
5.4	empiler()	6
5.4.1	Traces	6
5.5	depiler()	7
5.5.1	Traces	8
5.6	redimensionnerPile()	9
5.7	libererPile()	9
5.8	afficherPile()	9
5.9	affichePileInt()	9
5.10	afficherPileChar()	9
5.11	afficherPileChaineChar()	10
6	File	11
6.1	initFile()	11
6.2	estVideFile()	11
6.3	estPleineFile()	11
6.4	enfiler()	11
6.4.1	Traces	12
6.5	defiler()	13
6.5.1	Traces	14
6.6	redimensionnerFile()	15
6.7	libererFile()	15
6.8	afficherFile()	15
6.9	afficherFileInt()	16
6.10	afficherFileChar()	16
6.11	afficherFileChaineChar()	16
7	Main	17

8	TestPile	18
8.1	testUnitairePile()	18
8.2	testInitPile()	18
8.3	testEstVidePile()	18
8.4	testEstPleinePile()	18
8.5	testEmpiler()	18
8.6	testDepiler()	18
9	TestFile	19
9.1	testUnitaireFile()	19
9.2	testInitFile()	19
9.3	testFileEstVide()	19
9.4	testFileEstPleine()	19
9.5	testEnfiler()	19
9.6	testDefiler()	19
10	Cas d'exécution	20
10.1	Pile	20
10.1.1	Initialisation pile	20
10.1.2	Pile est vide	20
10.1.3	Pile est pleine	20
10.1.4	Empiler	20
10.1.5	Dépiler	20
10.2	File	21
10.2.1	Initialisation file	21
10.2.2	File est vide	21
10.2.3	File est pleine	21
10.2.4	Enfiler	21
10.2.5	Défiler	21
11	Annexe	22
11.1	Pile	22
11.1.1	initPile	22
11.1.2	estVidePile & estPleinePile	22
11.1.3	empiler	23
11.1.4	depiler	23
11.1.5	redimensionnerPile	24
11.1.6	afficherPile	24
11.2	File	26
11.2.1	initFile	26
11.2.2	estVideFile & estPleineFile	26
11.2.3	enfiler	27
11.2.4	defiler	28
11.2.5	redimensionnerFile	28
11.2.6	afficherPile	29
11.3	Main	31

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.

```
bin
└─ executable
makefile
src
├─ file.c
├─ file.h
├─ main.c
├─ pile.c
├─ pile.h
├─ testFile.c
├─ testFile.h
├─ testPile.c
└─ testPile.h
```

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'exécutable 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()

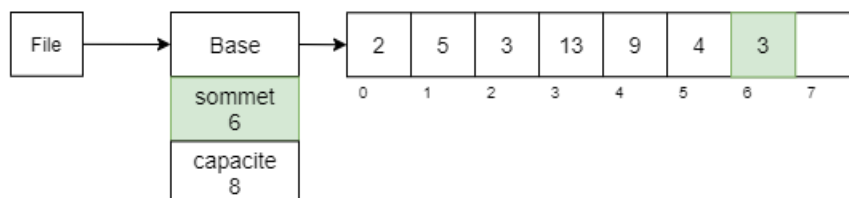
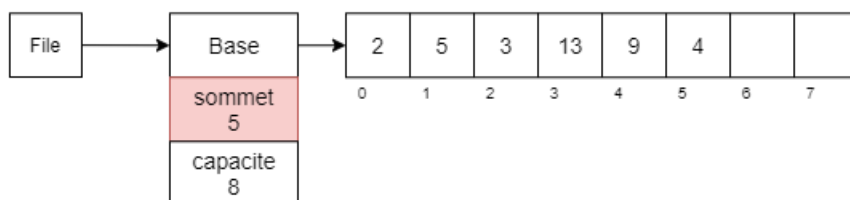
Empile une valeur à la fin de la pile, et l'agrandit si elle est trop petite.

5.4.1 Traces

Empiler

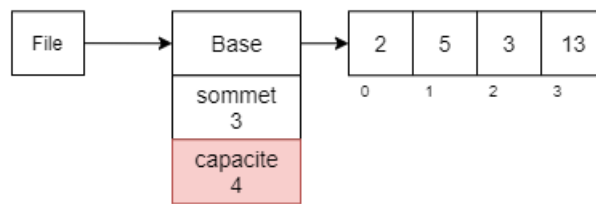
Cas général, on empile une valeur

Valeur = 3

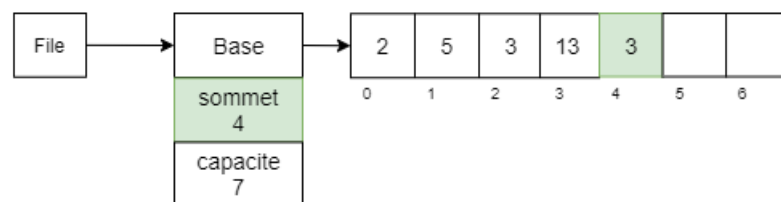
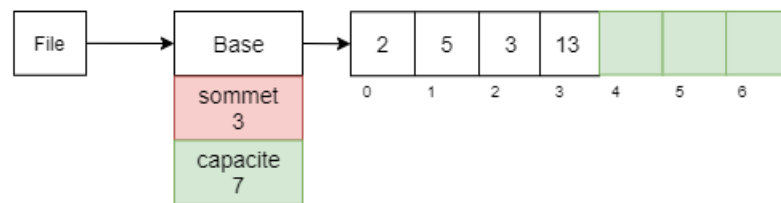


Empiler On empile une valeur dans une pile pleine, la pile est redimensionnée

Valeur = 3



On redimensionne la pile



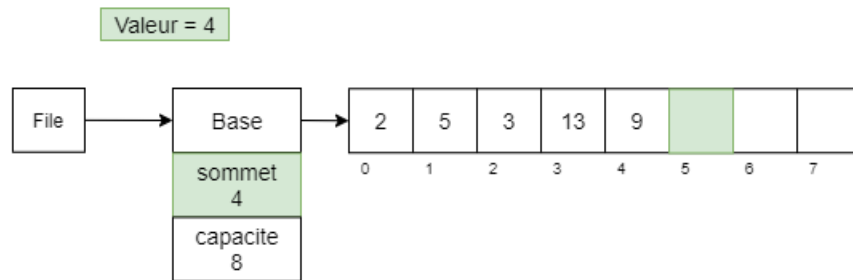
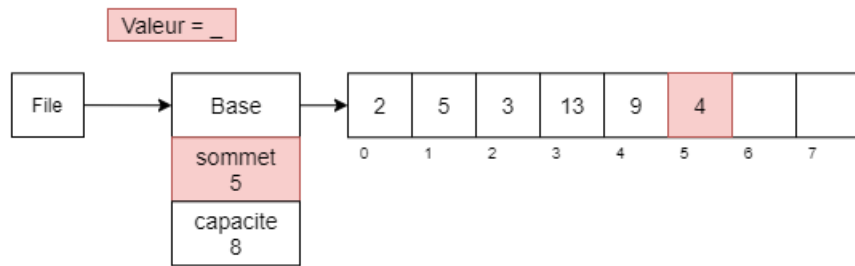
5.5 depiler()

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

5.5.1 Traces

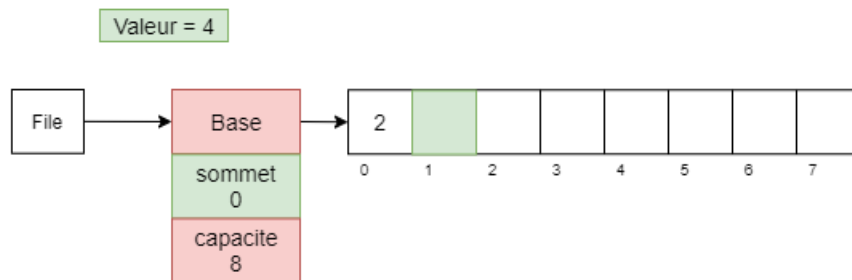
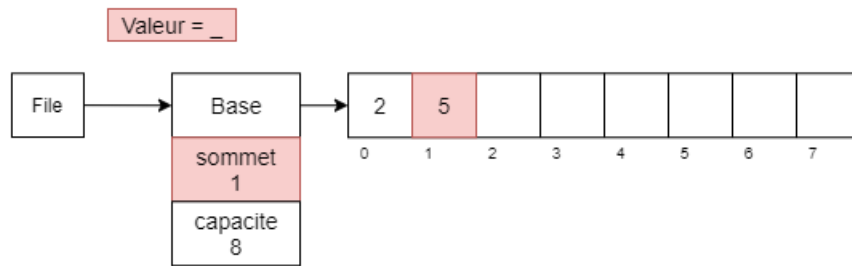
Depiler

Cas général, on depile une valeur

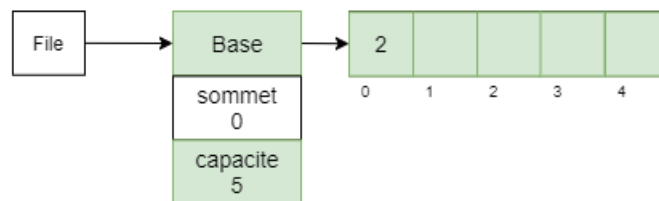


Depiler

On depile une valeur d'une pile peu utilisée, puis la pile est redimensionnée



On redimensionne la pile
Valeur = 4



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 afficherPileInt()

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()

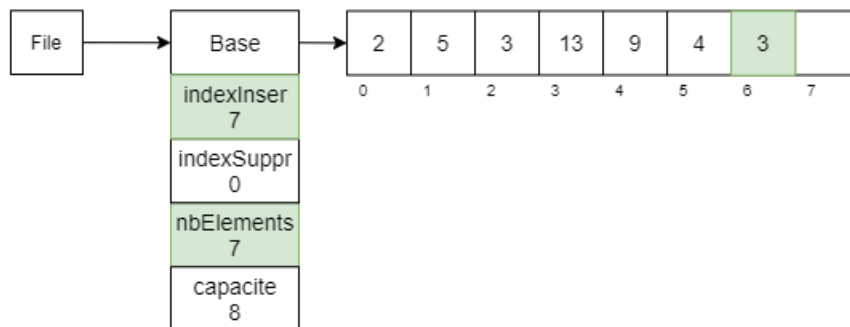
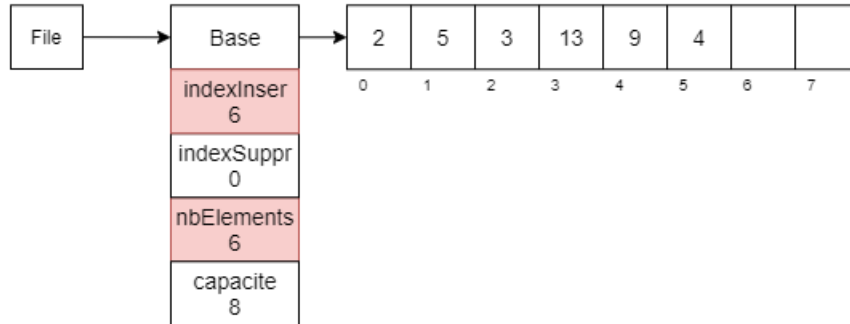
Enfile une valeur à la fin de la file, et l'agrandit si elle est trop petite.

6.4.1 Traces

Enfiler

Cas général, on enfile une valeur

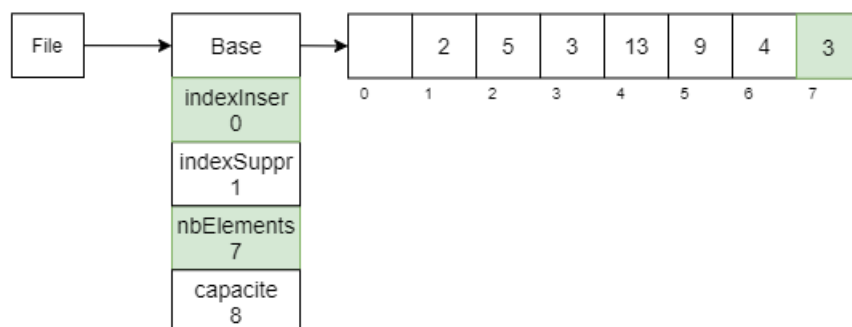
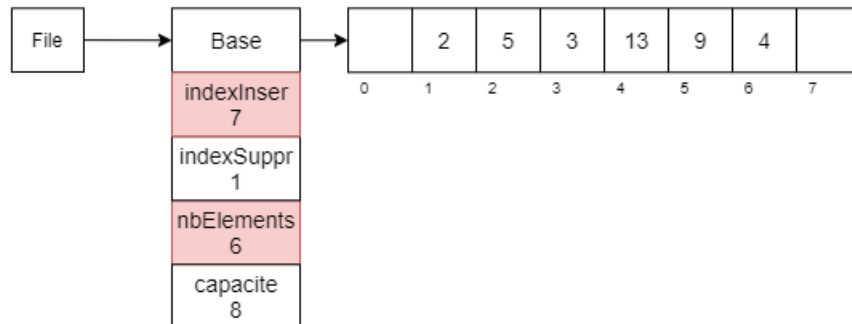
Valeur = 3



Enfiler

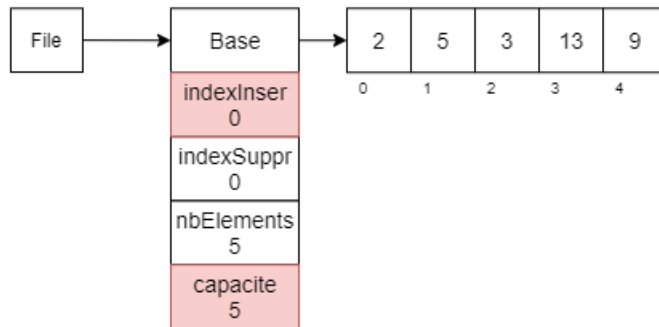
On enfile une valeur, l'indexInsertion repasse au début de la liste contigue

Valeur = 3

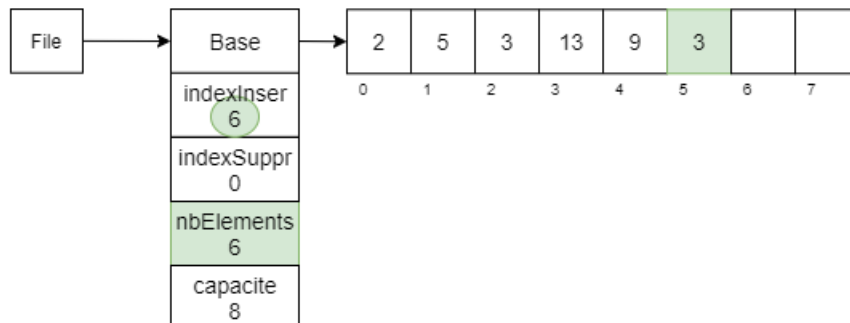
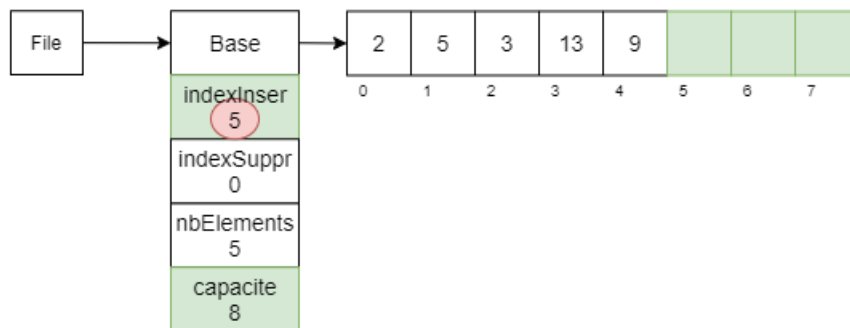


Enfiler On enfiler dans une file pleine, donc redimensionnement

Valeur = 3



La file est redimensionnée



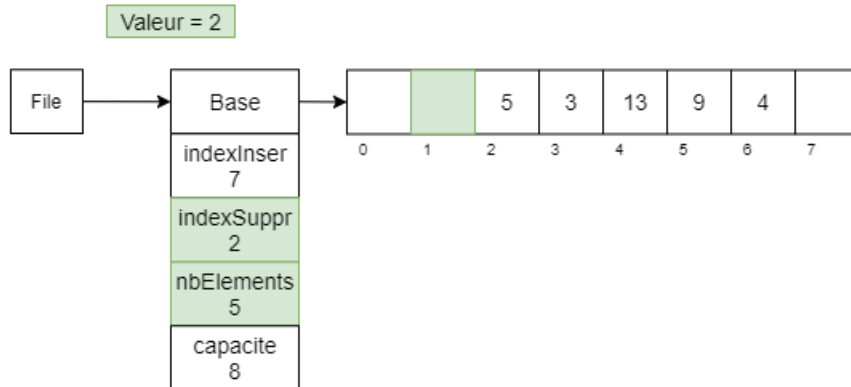
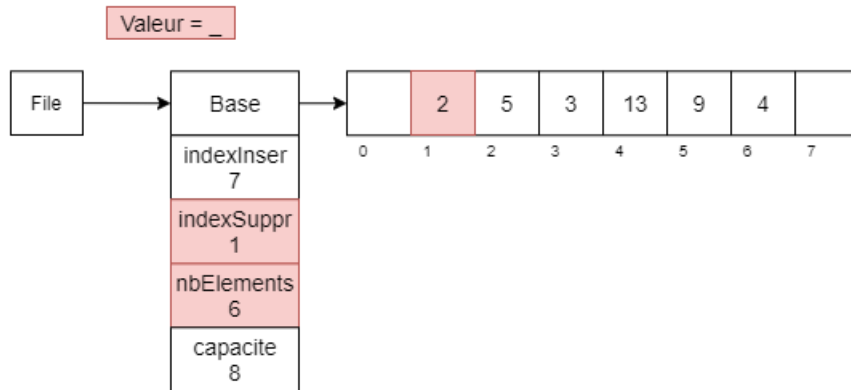
6.5 defiler()

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

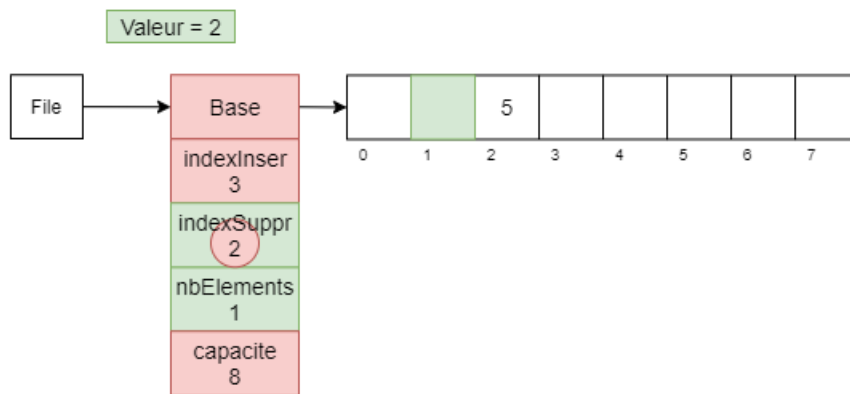
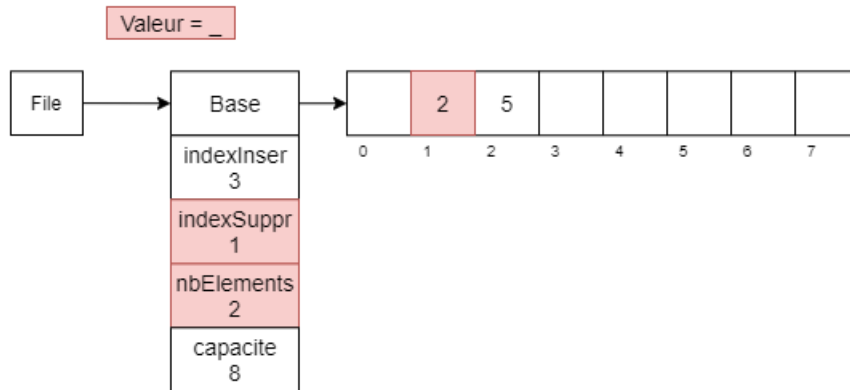
6.5.1 Traces

Defiler

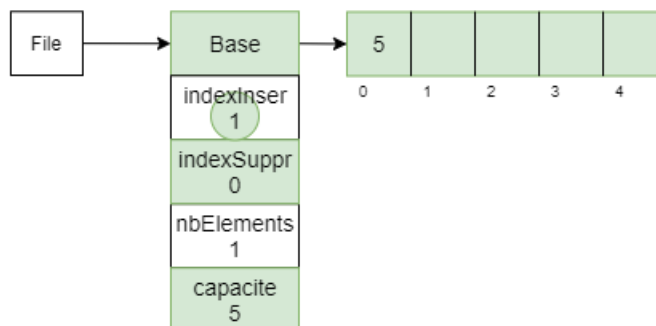
Cas général, on defile une valeur



Defiler On defile une valeur d'une file peu utilisée, puis la file est redimensionnée



La file est redimensionnée
Valeur = 2



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 fonctionnent. Chaque fonction de ce fichier teste une unique manipulation.

8.1 testUnitairePile()

Fonction appelant toutes les fonctions de test de la pile.

8.2 testInitPile()

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

8.3 testEstVidePile()

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

8.4 testEstPleinePile()

Fonction qui teste la fonction testant si la pile est pleine.

8.5 testEmpiler()

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

8.6 testDepiler()

Fonction qui teste la fonction de dépilement de la pile.

9 TestFile

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

9.1 testUnitaireFile()

Fonction appelant toutes les fonctions de test de la file.

9.2 testInitFile()

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

9.3 testFileEstVide()

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

9.4 testFileEstPleine()

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

9.5 testEnfiler()

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

9.6 testDefiler()

Fonction qui teste 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 créée
- 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 créée
- 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 créée
- 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 créée
- La pile est partiellement pleine
- La pile est pleine
- La pile contient 1 élément et on doit redimensionner
- La pile est partiellement pleine et on doit redimensionner
- 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 créée
- 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 créée
- 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 créée
- 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 créée
- 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 *);
```

```

44 int estVidePile(pile_t * pile) {
45     return (*pile).sommet == -1;
46 }
47
48
49 int estPleinePile(pile_t * pile) {
50     return (*pile).capacite == (*pile).sommet + 1;
51 }

```

11.1.3 empiler

```

52 /* ----- */
53 /* empiler      Empile une valeur au sommet de la pile,      */
54 /*              et l'aggrandit si elle est trop petite      */
55 /* ----- */
56 /* En entrée :  pile (pile_t *) : Pile à remplir            */
57 /*              valeur (type)   : Élément à empiler         */
58 /* ----- */
59 /* En sortie :  codeErreur (int) booléen : 1 si erreur, 0 sinon */
60 /* ----- */
61 int empiler(pile_t *, type);

```

```

54 int empiler(pile_t * pile, type v) {
55     int codeErreur = 1;
56
57     /* Si la pile est pleine, on redimensionne, sinon, on empile */
58     if (estPleinePile(pile)) {
59
60         /* + 1 pour le cas où la capacité vaut 1 */
61         int nvCapacite = 1.5 * (*pile).capacite + 1;
62
63         /* Si le redimensionnement a fonctionné, on empile la valeur */
64         if (!redimensionnerPile(pile, nvCapacite))
65             codeErreur = empiler(pile, v);
66
67     } else {
68         codeErreur = 0;
69         (*pile).sommet ++;
70         (*pile).base [(*pile).sommet] = v;
71     }
72
73     return codeErreur;
74 }

```

11.1.4 depiler

```

63 /* ----- */
64 /* depiler      Dépile la valeur au sommet de la pile,      */
65 /*              et la réduit si elle peu utilisée           */
66 /* ----- */
67 /* En entrée :  pile (pile_t *) : Pile à vider              */
68 /*              valeur (type)   : Élément dépilé            */
69 /* ----- */
70 /* En sortie :  codeErreur (int) booléen : 2 si erreur, 0 sinon */
71 /* ----- */
72 int depiler(pile_t *, type *);

```

```

77 int depiler(pile_t * pile, type * v) {
78     int codeErreur = 1;
79
80     /* Si la pile est non vide, on depile */
81     if (!estVidePile(pile)) {
82         codeErreur = 0;
83         *v = (*pile).base [(*pile).sommet];
84         (*pile).sommet --;
85
86         /* Si la pile est tres peu utilise, on diminue la taille */
87         if ((*pile).sommet < (int)0.25*(*pile).capacite) {
88             int nvCapacite = 0.5*(*pile).capacite + 1;
89             redimensionnerPile(pile, nvCapacite);
90         }
91     }
92
93     return codeErreur;
94 }

```

11.1.5 redimensionnerPile

```

74 /* ----- */
75 /* redimensionnerPile   Redimensionne la base de la pile      */
76 /* ----- */
77 /* En entrée : pile (pile_t *) : Pile à redimensionner        */
78 /*             nvCapacité (int) : Taille de la nouvelle base   */
79 /* ----- */
80 /* En sortie : codeErreur (int) booléen : 1 si erreur, 0 sinon */
81 /* ----- */
82 int redimensionnerPile(pile_t *, int);

```

```

97 int redimensionnerPile(pile_t * pile, int nvCapacite) {
98     int codeErreur = 1;
99
100    /* On realloc la base */
101    type * nvBase = realloc((*pile).base, nvCapacite*sizeof(type));
102    if (nvBase != NULL) {
103        codeErreur = 0;
104        /* Si l'allocation c'est bien faite, on modifie la base et la capacite */
105        (*pile).base = nvBase;
106        (*pile).capacite = nvCapacite;
107    }
108    return codeErreur;
109 }

```

11.1.6 afficherPile

```

93 /* ----- */
94 /* afficherPile        Affiche la pile                        */
95 /* ----- */
96 /* En entrée : pile (pile_t *) : Pile à afficher              */
97 /*             pfAfficher (void (*)(type)) : pointeur sur     */
98 /*             la fonction d'affichage du bon type             */
99 /* En sortie : void                                            */
100 /* ----- */
101 void afficherPile(pile_t *, void (*) (type));

```



```

104  /* ----- */
105  /* afficherPileInt      Affiche un élément de type int */
106  /* ----- */
107  /* En entrée : nombre (int) : Nombre à afficher */
108  /* ----- */
109  /* En sortie : void */
110  /* ----- */
111  void afficherPileInt(int);
112
113  /* ----- */
114  /* afficherPileChar    Affiche un élément de type char */
115  /* ----- */
116  /* En entrée : caractere (char) : Caractère à afficher */
117  /* ----- */
118  /* En sortie : void */
119  /* ----- */
120  void afficherPileChar(char);
121
122  /* ----- */
123  /* afficherPileChaineCarac    Affiche un élément de type chaine de caractère */
124  /* ----- */
125  /* En entrée : chaine (char *) : Chaine de caractères à afficher */
126  /* ----- */
127  /* En sortie : void */
128  /* ----- */
129  void afficherPileChaineCarac(char *);
130

```

```

112  void afficherPile(pile_t * pile, void (*pfAfficher) (type)) {
113
114      printf("Pile : capacite=%d\n", (*pile).capacite);
115      printf("      ");
116
117      if(!estVidePile(pile)) {
118          /* On affiche chaque element un par un en appelant la fonction d'affichage */
119          for (int i=0; i <=(*pile).sommet; i++)
120              (*pfAfficher) ( (*pile).base[i] );
121      } else {
122          printf("vide");
123      }
124
125      printf("\n");
126  }
127
128
129  void afficherPileInt(int nombre) {
130      printf("%d ", nombre);
131  }
132
133
134  void afficherPileChar(char caractere) {
135      printf("%c ", caractere);
136  }
137
138
139  void afficherPileChaineCarac(char * chaine) {
140      printf("%s ", chaine);
141  }

```

11.2 File

11.2.1 initFile

```
27  /* ----- */
28  /* initFile   Initialise une file de taille le paramètre taille */
29  /* ----- */
30  /* En entrée : file : file_t * : File à tester */
31  /* ----- */
32  /* En sortie : file : Pointeur sur la file (file_t *) */
33  /* ----- */
34  file_t * initFile(int);
```

```
11 ▾ file_t * initFile(int capacite) {
12     file_t * file = (file_t *)malloc(sizeof(file_t)); /*File créée*/
13
14     if (file != NULL) {
15         file->base = (type *)malloc(capacite * sizeof(capacite));
16
17         /* Erreur de memoire */
18         if (file->base == NULL) {
19             free(file);
20             file = NULL;
21
22             /* Initialisation de la file */
23         } else {
24             file->capacite      = capacite;
25             file->nbElements    = 0;
26             file->indexSuppression = 0;
27             file->indexInsertion  = 0;
28         }
29     }
30
31     return file;
32 }
```

11.2.2 estVideFile & estPleineFile

```
37  /* ----- */
38  /* estVideFile Indique si la file passée en paramètre est vide ou non */
39  /* ----- */
40  /* En entrée : file (file_t *) : File à tester */
41  /* ----- */
42  /* En sortie : (char) booléen : 1 si la file est vide, 0 sinon */
43  /* ----- */
44  char estVideFile(file_t *);
```

```
47  /* ----- */
48  /* estPleineFile Indique si la file passée en paramètre est pleine ou non */
49  /* ----- */
50  /* En entrée : file (file_t *) : File à tester */
51  /* ----- */
52  /* En sortie : (char) booléen : 1 si la file est pleine, 0 sinon */
53  /* ----- */
54  char estPleineFile(file_t * file);
```

```

35 char estVideFile(file_t * file) {
36     return (file->nbElements == 0);
37 }
38
39
40 char estPleineFile(file_t * file) {
41     return (file->nbElements == file->capacite);
42 }

```

11.2.3 enfiler

```

57 /* ----- */
58 /* enfiler      Enfile une valeur à la fin de la file,      */
59 /*              et l'aggrandit si elle est trop petite      */
60 /* ----- */
61 /* En entrée :   file (file_t *) : File à remplir          */
62 /*              valeur (int)      : Valeur à enfiler        */
63 /* ----- */
64 /* En sortie :   codeErreur (char) booléen : 1 si erreur, 0 sinon */
65 /* ----- */
66 char enfiler(file_t * file, type valeur);

```

```

45 char enfiler(file_t * file, type valeur) {
46     char codeErreur = 1; /*Code erreur*/
47
48     /* Si la file n'est pas déjà remplie, on enfile la valeur */
49     if (!estPleineFile(file)) {
50         file->base[file->indexInsertion] = valeur;
51         file->indexInsertion = (file->indexInsertion + 1) % file->capacite; /*Index d'insertion suivante*/
52
53         file->nbElements += 1;
54         codeErreur = 0;
55
56         /* Sinon, il faut redimensionner la file */
57     } else {
58         int nouvCapacite = 1.5 * file->capacite + 1;
59
60         /* Si la redimension est sans erreur, on enfile la valeur */
61         if (redimensionnerFile(file, nouvCapacite)) {
62             printf("Redimensionnement impossible\n");
63             codeErreur = 1;
64
65         } else {
66             enfiler(file, valeur);
67             codeErreur = 0;
68         }
69     }
70
71     return codeErreur;
72 }

```

11.2.4 defiler

```
69  /* ----- */
70  /* defiler      Défile la valeur en tête de la file,      */
71  /*              et la réduit si elle peu utilisée         */
72  /* ----- */
73  /* En entrée :  file (file_t *) : File à vider           */
74  /*              valeur (int)    : Valeur dépiquée (par adresse) */
75  /* ----- */
76  /* En sortie :  codeErreur (char) booléen : 2 si erreur, 0 sinon */
77  /* ----- */
78  char defiler(file_t * file, type * valeur);
79
75  char defiler(file_t * file, type * valeur) {
76      char codeErreur = 2; /*Code erreur*/
77
78      /* Si la file n'est pas vide, on defile la valeur */
79      if (!estVideFile(file)) {
80          *valeur = file->base[file->indexSuppression];
81          file->indexSuppression = (file->indexSuppression + 1) % file->capacite;
82          file->nbElements -= 1;
83          codeErreur = 0;
84
85          /* Si la file est peu utilisée, on la redimensionne */
86          if (file->nbElements < 0.25 * file->capacite) {
87              int nouvCapacite = 0.5 * file->capacite + 1;
88
89              if (redimensionnerFile(file, nouvCapacite)) {
90                  printf("Redimensionnement impossible\n");
91                  codeErreur = 2;
92              }
93          }
94      } else {
95          printf("File vide\n");
96      }
97
98      return codeErreur;
99  }
100 }
```

11.2.5 redimensionnerFile

```
81  /* ----- */
82  /* redimensionnerFile Redimensionne la file en copiant les valeurs */
83  /* ----- */
84  /* En entrée :  file (file_t *) : File à redimensionner      */
85  /*              nouvCapacite (int) : Taille de la nouvelle file */
86  /* ----- */
87  /* En sortie :  codeErreur (char) booléen : 1 si erreur, 0 sinon */
88  /* ----- */
89  char redimensionnerFile(file_t * file, int nouvCapacite);
```

```

103 char redimensionnerFile(file_t * file, int nouvCapacite) {
104     char codeErreur = 1; /*Code Erreur*/
105     int i = 0; /*Compteur*/
106
107     type * nouvBase = (type *)malloc(nouvCapacite * sizeof(type));
108
109     /* Si l'allocation est sans erreur, on copie la file */
110     if (nouvBase != NULL) {
111         /* On copie l'ancienne file dans la nouvelle (redimensionnée) */
112         for (i=0; i<file->nbElements; i++) {
113             nouvBase[i] = file->base[(file->indexSuppression + i) % file->capacite];
114         }
115
116         /* On libère l'ancienne file et on actualise les données de la nouvelle */
117         free(file->base);
118         file->base = nouvBase;
119
120         /* On actualise les données de la file */
121         file->indexSuppression = 0;
122         file->indexInsertion = file->nbElements;
123         file->capacite = nouvCapacite;
124
125         codeErreur = 0;
126     }
127     return codeErreur;
128 }

```

11.2.6 afficherPile

```

102 /* ----- */
103 /* afficherFile Affiche la file passée en paramètre */
104 /* ----- */
105 /* En entrée : file (file_t *) : File à afficher */
106 /*             pfAfficher (void (*)(type)) : pointeur sur */
107 /*             la fonction d'affichage du bon type */
108 /* ----- */
109 /* En sortie : void */
110 /* ----- */
111 void afficherFile(file_t * file, void (*pfAfficher) (type));

```

```

114  /* ----- */
115  /* afficherFileInt  Affiche un entier */
116  /* ----- */
117  /* En entrée :  nombre (int) : int à afficher */
118  /* ----- */
119  /* En sortie :  void */
120  /* ----- */
121  void afficherFileInt(int nombre);
122
123
124  /* ----- */
125  /* afficherFileChar  Affiche un caractère */
126  /* ----- */
127  /* En entrée :  file (char) : char à afficher */
128  /* ----- */
129  /* En sortie :  void */
130  /* ----- */
131  void afficherFileChar(char caractere);
132
133
134  /* ----- */
135  /* afficherFileChaineChar  Affiche une chaine de caractères */
136  /* ----- */
137  /* En entrée :  chaine (char *) : Chaine à afficher */
138  /* ----- */
139  /* En sortie :  void */
140  /* ----- */
141  void afficherFileChaineChar(char * chaine);

```

```

138  void afficherFile(file_t * file, void (*pfAfficher) (type)) {
139      int i = 0; /*Compteur*/
140
141      if (!estVideFile(file)) {
142          for (i=0; i<file->nbElements; i++) {
143              pfAfficher(file->base[(file->indexSuppression + i) % file->capacite]);
144          }
145
146      } else {
147          printf("File vide\n");
148      }
149  }
150
151
152  void afficherFileInt(int nombre) {
153      printf("%d\n", nombre);
154  }
155
156
157  void afficherFileChar(char caractere) {
158      printf("%c\n", caractere);
159  }
160
161
162  void afficherFileChaineChar(char * chaine) {
163      printf("%s\n", chaine);
164  }

```

11.3 Main

```
50 int main(int argc, char const *argv[]) {
51
52     if (argc >= 2 && *argv[1] == '1') {
53
54         if (testUnitairePile() == 1)
55             printf("\033[33m    Fonctions pile \033[32mfonctionne\033[00m\n\n");
56         else
57             printf("\033[33m    Fonctions pile \033[31mne fonctionne pas\033[00m\n\n");
58
59         if (testUnitaireFile())
60             printf("\033[33m    Fonctions file \033[32mfonctionne\033[00m\n\n");
61         else
62             printf("\033[33m    Fonctions file \033[31mne fonctionne pas\033[00m\n\n");
63
64     } else {
65
66         int n = 26;
67         pile_t * pile = initPile(10);
68
69         for (int i=0; i<n; i++)
70             empiler(pile, 97+i);
71
72         afficherPile(pile, afficherPileChar);
73         if (!inverserPile(pile))
74             afficherPile(pile, afficherPileChar);
75         else
76             printf("Erreur lors de l'inversion de la pile\n");
77
78         libererPile(pile);
79     }
80
81     return 0;
82 }
83 }
```