TP 2 Structure de données

Monteilhet Gautier
Champredonde Adrien

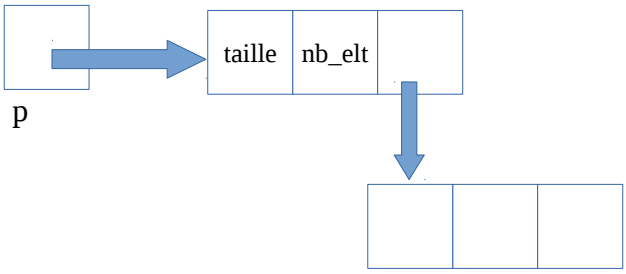
1) Présentation générale

Objectif :

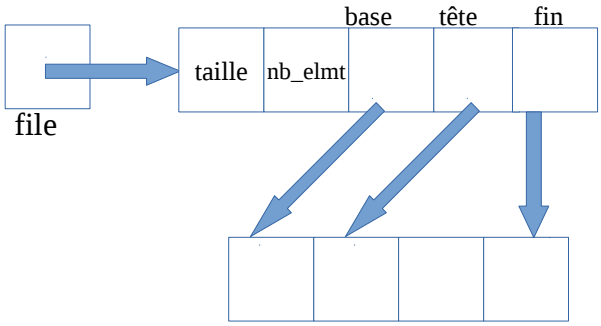
Ce tp a pour objectif de mettre en place la gestion d'une pile et d'une file. Tout en créant les structures de celles-ci.

Structure de données :

Une pile



Une file



Organisation du code source :

pile.h Structure de la pile et déclaration des prototypes des fonctions de gestion de la pile.

pile.c Fonctions : - InitPile
- Pleine_Pile
- Vide_Pile
- Empiler
- Depiler
- LibererPile
- AfficherPile

file.h Structure de la file et déclaration des prototypes des fonctions de gestion de la file.

file.c Fonctions : - InitFile
- Pleine_File
- Vide_File
- Enfiler
- Defiler
- LibererFile
- AfficherFile

tp2.h Prototype de la fonction InverseEntierPile

tp2.c Fonction : InverseEntierPile

main.c Fonction main.

2) Présentation de chaque fonction

pile.c

fonction InitPile(E : n)
    On alloue l'espace mémoire nécessaire pour créer une pile p;
    Si l'allocation n'a pas fonctionné alors
        afficher 'Problème d'allocation pour la pile';
        quitter le programme ;
    fsi ;
    Initialiser la taille de la pile à n ;
    Initialiser le nombre d'élément de la pile à 0 ;
    On alloue de l'espace mémoire pour créer un tableau avec une taille correspond à celle de la pile ;
    Si l'allocation n'a pas fonctionné alors
        afficher 'Problème d'allocation pour la pile';
        On libère l'espace occupé par la pile ;
        quitter le programme ;
    fsi ;
    retourner p ;
Fin ;

fonction Pleine_Pile(E : p)
    retourner si la taille de la pile est égale au nombre d'élément ou non;
Fin;

fonction Vide_Pile(E : p)
    retourner si le nombre d'élément de la pile est égale à 0 ou non;
Fin ;

fonction Empiler(E/S : p ; E: val)
    On initialise le code d'erreur à 1 qui signifie aucune erreur ;
    Si la pile n'est pas pleine alors
        On ajoute un élément de plus à la pile ;
        On place le nouvel élément à la fin de notre tableau de valeurs ;
    Sinon
        On affiche que la pile est pleine ;
        La variable erreur est égale à 0 (erreur présente);
    fsi ;
    retourner la variable erreur ;
Fin ;

fonction Depiler(E/S : p ; S: val)
	On initialise le code d'erreur à 1 qui signifie aucune erreur ;
	Si la pile n'est pas vide alors
		Le pointeur val pointe sur l'élément retirer ;
		On décrémente le nombre d'élément dans la pile ;
	Sinon
		On affiche que la pile est vide ;
		La variable erreur est égale à 0 (erreur présente);
	fsi ;
	retourner la variable erreur ;
Fin ;

procédure LibererPile(E : p)
	Si la pile existe alors
		Si le tableau de valeur existe alors
			On libère le tableau des valeurs ;
		fsi ;
		On libère la structure pile ;
	fsi ;
Fin ;

procédure AfficherPile(E : p)
	On affiche chaque élément du tableau des valeurs de la pile ;
Fin ;

file.c

fonction InitFile(E : n)
	On alloue l'espace mémoire nécessaire pour créer une file;
	Si l'allocation n'a pas fonctionné alors
		afficher 'Problème d'allocation pour la file';
		quitter le programme ;
	fsi ;
	Initialiser la taille de la file à n ;
	Initialiser le nombre d'élément de la file à 0 ;
	On alloue de l'espace mémoire pour créer un tableau avec une taille correspond à celle de la file. Ce sera le pointeur base de la file qui pointera sur cet espace ;
	Si l'allocation n'a pas fonctionné alors
		afficher 'Problème d'allocation pour la file';
		On libère l'espace occupé par la file ;
		quitter le programme ;
	fsi ;
	Le pointeur tete de la file pointe sur la tête du tableau de valeurs.
	Le pointeur fin de la file pointe sur la fin du tableau de valeurs.
	retourner la file ;
Fin ;

```
fonction Pleine_File(E : file)
        retourner si la taille de la file est égale au nombre d'élément ou non;
Fin;


fonction Vide_File(E : file)
        retourner si le nombre d'élément de la file est égale à 0 ou non;
Fin ;


fonction Enfiler(E/S : file ; E: e)
        On initialise le code d'erreur à 1 qui signifie aucune erreur ;
        Si la file n'est pas pleine alors
                On décale le pointeur de fin d'une case mémoire ;
                On ajoute un élément de plus à la file avec le pointeur de fin ;
                On incrémente le nombre d'élément dans la file;
        Sinon
                On affiche que la file est pleine ;
                La variable erreur est égale à 0 (erreur présente);
        fsi ;
        retourner la variable erreur ;
Fin ;


fonction Defiler(E/S : file ; S: e)
        On initialise le code d'erreur à 1 qui signifie aucune erreur ;
        Si la file n'est pas vide alors
                La valeur de e est égale à la valeur pointé par le pointeur de tete ;
                On décale le pointeur de tete d'une case mémoire ;
                On décrémente le nombre d'élément dans la file ;
        Sinon
                On affiche que la file est vide ;
                La variable erreur est égale à 0 (erreur présente);
        fsi ;
        retourner la variable erreur ;
Fin ;


procédure LibererFile(E : file)
        Si la file existe alors
                Si le tableau de valeur pointé par le pointeur base existe alors
                        On libère le tableau des valeurs ;
                fsi ;
                On libère la structure file ;
        fsi ;
Fin ;
```

```
procédure AfficherFile(E : file)
        Si la file est vide alors
                On affiche la file est vide
        On affiche chaque élément du tableau des valeurs de la file ;
Fin ;
```

tp2.c

```
fonction InverseEntierPile(E/S : pile, file)
        On initialise j au nombre d'élément dans la pile ;
        Pour chaque élément de la pile faire
                Depiler de la pile ;
                Si aucune erreur alors
                        Enfiler dans la file ;
                fsi ;
        ftq ;
        Si aucune erreur alors
                On initialise j au nombre d'élément dans la file ;
                Pour chaque élément de la file faire
                        Si aucune erreur alors
                                Defiler de la file ;
                                Si aucune erreur alors
                                        Empiler sur la pile ;
                                fsi ;
                        fsi ;
                ftq ;
        fsi ;
        retourner le code erreur;
Fin ;
```

main.c

```
fonction main()
        Initialisation de la pile avec une taille de 5 ;
        Initialisation de la file avec une taille de 5 ;
        On empile le chiffre 4 ;
        On empile le chiffre 2 ;
        On empile le nombre 79 ;
        On affiche la pile ;
        Si il n'y a pas d'erreur alors
                On inverse les entiers de la pile ;
        fsi ;
        On affiche la pile ;
        On libère la file ;
        On libère la pile ;
        On retourne le code erreur ;
Fin ;
```

3) Compte rendu d'exécution

Les cas particuliers : - pile vide
- pile pleine
- file plus petite que la pile
- pile plus petite que la file

Pile vide :

```
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/tp2$ ./prog

Pile :


Pile :
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/tp2$
```

Pile pleine :

```
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/tp2$ ./prog

La pile est pleine !

Pile :
2
4


Pile :
2
4
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/tp2$
```

Pile plus grande que la file :

```
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/tp2$ ./prog

Pile :
79
2
4
File pleine !
File pleine !


Pile :
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/tp2$
```

Pile plus petite que la file :

```
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/tp2$ ./prog

Pile :
79
2


Pile :
2
79
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/tp2$
```