

Structure de données

**Compte rendu :** Listes chainées/

Les piles /Les files

Filière d’ingénieur:

**I**ngénierie **L**ogicielle et **I**ntégration des **S**ystèmes **I**nformatiques

Encadré par :

* Pr. A. BEKKHOUCHA

Réalisé par :

* SOUNEINI Salma
* ANEBA Meriem

1

Année universitaire : 2023-2024

**Les files :**

* Définition des fonctions de manipulation des files mise

en œuvre à l’aide d’un pointeur **:**

#include <stdio.h>

#include <stdlib.h>

// Définition de la structure de la cellule

typedef struct Cellule

{

int entier;

struct Cellule \*svt; // Pointeur vers la cellule suivante

} Cellule;

// Définition de la structure de la file

typedef struct File

{

Cellule \*debut; // Pointeur vers le début de la file

Cellule \*fin; // Pointeur vers la fin de la file

} PTFile;

//fonction d'allocation d'espace memoire pour la ma file

PTFile\* creer\_Ptfile()

{

PTFile \*NF;

NF=(PTFile \*)malloc(sizeof(PTFile));//allocation de l'espace mémoire

// si l'allocation a echoué

if(!NF)

{

printf("\nErreur d'allocation");

exit(1); // si l'allocation a echoué

}

NF->debut=NF->fin=NULL;//initialiser les éléments de la structue PTFile

return ((PTFile\*)NF);//retourne l'adresse de l'espace mémoire réservé

}

//creer une cellule

Cellule\* creer\_Cellule()

{

Cellule \*Cl;

Cl=(Cellule \*)malloc(sizeof(Cellule));//allocation de l'espace mémoire

// si l'allocation a echoué

if(!Cl)

{

printf("\nErreur d'allocation");

exit(1); // si l'allocation a echoué

}

Cl->svt=NULL;//initialiser les elements de la structue PTFile

return ((Cellule\*)Cl);//retourne l'adresse de l'espace mémoire réservé

}

// Fonction pour vérifier si la file est vide

int EstVide\_File(PTFile \*Mafile)

{

return((int)!Mafile->debut);

}

// Fonction pour vider la file

int Razer\_PTfile(PTFile \*Mafile)

{

PTFile \*tmp;

//teste d'existance

if(!Mafile)

{

printf("\nerror d'allocation "); return ((int)0);

}

//supprimer tous les éléments de la file

while(Mafile->debut)

{

tmp=Mafile;//tmp pointe sur la cellule pointé par Mafile

//Mafile pointe sur la cellule suivante

Mafile->debut=Mafile->debut->svt;

free(tmp);//liberer l'espace mémoire pointé par tmp

}

/\*affecter NULL au pointeur fin pour s'assurer que fin ne

contient aucune adresse que Null\*/

Mafile->fin=NULL;

return ((int)1);

}

// Fonction pour enfiler une cellule dans la file

int enfiler(PTFile \*Mafile,Cellule \*NF)

{

//si Mafile ou NF n'existe pas

if(!Mafile || !NF)

{

/\*erreur d'allocation de l'espace mémoire

ou bien la cellule n'existe pas\*/

printf("n erreur d'allocation");

return ((int)0);

}

//si Mafile existe mais il est vide

if(EstVide\_File(Mafile))

{

/\*La cellule pointé par début et fin va

être la 1ere cellule \*/

Mafile->debut=Mafile->fin=NF;

}

//si la file n'est pas vide

Mafile->fin->svt=NF;

Mafile->fin=NF;

return((int)1);

}

// Fonction pour défiler une cellule de la file

int defiler(PTFile \*Mafile)

{

Cellule \*tmp;

//si la file n'existe pas

if(!Mafile)

{

printf("\nerreur d'allocation e l'espace mémoire");

return ((int)-2);

}

//si la file existe mais il est vide

if(EstVide\_File(Mafile))

{

printf("\nla file existe mais il est vide");

return ((int)-1);

}

//si la file contient un seul element

if(Mafile->debut==Mafile->fin)

{

return ((int)Razer\_PTfile(Mafile));

}

//si la file contient plusieur cellule

tmp=Mafile->debut;//tmp pointe sur la cellule pointé par Mafile

//Mafile pointe sur la cellule suivant

Mafile->debut=Mafile->debut->svt;

free(tmp);//liberer l'espace mémoire pointe par tmp

return ((int)1);

}

// Fonction pour afficher tous les éléments de la file

void afficher(PTFile Mafile)

{

Cellule \*tmp;

//si la file est vide

if(EstVide\_File(&Mafile))

{

printf("\non peut pas afficher une file vide");

}

//si la file n'est pas vide

tmp=Mafile.debut;

while(tmp)

{

printf("%d\t",tmp->entier);

tmp=tmp->svt;

}

}

int main()

{

PTFile \*Mafile = creer\_Ptfile();

// Crée et initialise une file vide

printf("File cree et initialisee.\n");

// Ajoute des éléments à la file

Cellule \*cel1 = creer\_Cellule();

cel1->entier = 4;

enfiler(Mafile, cel1);

Cellule \*cel2 = creer\_Cellule();

cel2->entier = 5;

enfiler(Mafile, cel2);

Cellule \*cel3 = creer\_Cellule();

cel3->entier = 6;

enfiler(Mafile, cel3);

printf("Elements ajoutes a la file : 4, 5 et 6.\n");

// Affiche les éléments de la file

printf("File actuelle : \n");

afficher(\*Mafile);

// Vérifie si la file est vide

printf("\nLa file est %s.\n", EstVide\_File(Mafile) ? "vide" : "non vide");

// Vérifie si la file est saturée (dans ce cas, elle ne peut pas l'être)

printf("La file est non saturee.\n");

// Défile un élément de la file

defiler(Mafile);

printf("Element defile.\n");

// Affiche la file après le défilement

printf("File actuelle : \n");

afficher(\*Mafile);

// Libère la mémoire

Razer\_PTfile(Mafile);

free(Mafile);

return 0;

}

* Définition des fonctions de manipulation des files mise

en œuvre à l’aide d’un tableau **:**

**Les Listes chainées:**

* Définition des fonctions de manipulation d’une liste mise

en œuvre à l’aide d’un pointeur **:**

#include <stdio.h>

#include <stdlib.h>

typedef struct lst{

int Entier;

struct lst \*svt;

//struct lst \*prd;

}Cellule;

//creer un cellule

Cellule\* Creer\_cellule(){

Cellule \*NE;

NE=(Cellule \*)malloc(sizeof(Cellule));

if(!NE)

{

printf("probleme de l'allocation");

exit((int)-1);

}

NE->svt=NULL;

return((Cellule\*)NE);

}

//tester si la liste est vide

int Est\_vide\_liste(Cellule \*liste)

{

// if(!liste) return((int)1);//si la liste est vide

// return((int)0);//si la liste n'est pas vide

return ((int)!liste);

}

//fonction qui retourne la taille de la liste

int Taille\_liste(Cellule \*liste)

{

int taille=0;

Cellule \*tmp;

tmp=liste;

//tant que tmp!=NULL

while(tmp)

{

taille++;//incrementation de taille

tmp=tmp->svt;//pointer sur la cellule suivant

}

return((int)taille);//retourner la taille

}

/\*La fonction insererPos permet d'insérer une cellule au

début, à la fin ou à une position quelconque\*/

Cellule \*insererPos(Cellule \*liste,int pos,Cellule \*NE)

{

Cellule \*crt;

int compt=0;

if(!NE) return ((Cellule \*)liste);

//insertion au debut de la liste

if(pos==0)

{

//La nouvelle cellule devient la première cellule de la liste

NE->svt=liste;

return((Cellule \*)NE);

}

//si la position donnée est non valide

if((pos<0) || pos>Taille\_liste(liste))

{

printf("\nposition non valide");

return ((Cellule \*)liste);

}

//si la position valide

crt=liste;

while(compt<pos-1)

{

compt++;

crt=crt->svt;

}

//Le pointeur 'crt' pointe vers la cellule qui précède la position de 1.

NE->svt=crt->svt;

crt->svt=NE;

return((Cellule \*)liste);

}

/\*La fonction supprimerPos permet de supprimer une cellule au début, à

la fin ou à une position quelconque\*/

Cellule \*supprimerPos(Cellule \*liste,int pos)

{

Cellule \*crt,\*tmp;

int compt=0;

//Supprimer au début de la liste

if(pos==0)

{

//La 2ème cellule devient la première

crt=liste;

liste=liste->svt;

free(crt);

return((Cellule \*)liste);

}

//si la position donnée est non valide

if((pos<0) || (pos>Taille\_liste(liste)))

{

printf("\nposition non valide");

return ((Cellule \*)liste);

}

//si la position valide

crt=liste;

while(compt<pos-1)

{

compt++;//incrementation du compteur

crt=crt->svt;

}

//Le pointeur 'crt' pointe vers la cellule qui précède la position de 1.

tmp=crt->svt;//tmp pointe sur la cellule à supprimer

crt->svt=tmp->svt;//crt pointe sur le suivan de tmp

free(tmp);//supprimer et libérer l'espace mémoire

return((Cellule \*)liste);

}

//fonction qui affiche les elements de la liste

void afficher\_liste(Cellule \*liste)

{

Cellule \*tmp;

//si la liste est vide

if(!liste)

{

printf("\non peut pas afficher une liste vide");

}

//si la liste n'est pas vide

tmp=liste;

while(tmp)

{

//afficher le contenu de la cellule "entier" dont laquel tmp pointe

printf("%d\t",tmp->Entier);

tmp=tmp->svt;//pointer sur l'element suivant

}

}

//vider la liste

Cellule\* vider\_liste(Cellule \*liste)

{

Cellule \*tmp;

//vider la liste

while(liste)

{

tmp=liste;//tmp poite sur liste

liste=liste->svt;//liste pointe sur element suivant

free(tmp);//liberer l'espace pointé par tmp

}

/\*on peut retourner liste mais c'est mieu de retourner NULL

pour assurer que liste ne contient pas une adresse que null\*/

return((Cellule \*)NULL);

}

int main() {

Cellule \*liste = NULL; // Initialisation de la liste à NULL (vide)

// Création de quelques cellules pour tester l'insertion

Cellule \*cell1 = Creer\_cellule();

cell1->Entier = 1;

Cellule \*cell2 = Creer\_cellule();

cell2->Entier = 2;

Cellule \*cell3 = Creer\_cellule();

cell3->Entier = 3;

Cellule \*cell4= Creer\_cellule();

cell4->Entier = 5;

Cellule \*cell5= Creer\_cellule();

cell5->Entier = 11;

// Affichage de la liste initiale (devrait être vide)

printf("Liste initiale :\n");

afficher\_liste(liste);

// Insertion des cellules dans la liste à des positions différentes

liste = insererPos(liste,0,cell1); // Insérer cell1 au début

liste = insererPos(liste,1,cell2); // Insérer cell2 à la position 1 (milieu)

liste = insererPos(liste,2,cell3); // Insérer cell3 à la position 2

liste = insererPos(liste,3,cell4); // Insérer cell3 à la fin

liste = insererPos(liste,2,cell5); // Insérer cell3 à la position 2

// Affichage de la liste après insertion

printf("\nListe après insertion :\n");

afficher\_liste(liste);

// Supprimer un élément à la position donnée

liste = supprimerPos(liste,3);

// Afficher la liste après la suppression

printf("\nListe après suppression :\n");

afficher\_liste(liste);

// Vider la liste

liste = vider\_liste(liste);

// Affichage de la liste après vidage (devrait être vide à nouveau)

printf("\nListe après vidage :\n");

afficher\_liste(liste);

return 0;

}

* Définition des fonctions de manipulation d’une liste mise

en œuvre à l’aide d’un Tableau **:**

#include <stdio.h>

#include <stdlib.h>

#define MaxElm 100

//programme de manipulation d'une liste statique

//définition de la structure

typedef struct lst

{

int TabInt[MaxElm];//table des entiers de taille max 100

int nbElm;//nbr d'element de la table

}Liste;

//allocation de la mémoire

Liste \*Creer\_Liste()

{

Liste \*p;

p = (Liste \*)malloc(sizeof(Liste)); //allocation de l'espace mémoire

if (!p) exit(1); // si l'allocation a échoué

return ((Liste \*)p); //retourne l'adresse de l'espace mémoire réservé

}

//initialisation du tableau

int initListe(Liste \*MaListe)

{

//si MaListe n'existe pas

if (!MaListe)

{

printf("\nErreur d'allocation "); return ((int)0);

}

//s'il existe, on initialise le nombre d'éléments

MaListe->nbElm = 0;

return ((int)1);

}

//tester est-ce que la liste est vide

int EstVide(Liste MaListe)

{

return ((int)MaListe.nbElm == 0);

}

//tester est-ce que la liste est saturée

int EstSaturee(Liste MaListe)

{

return ((int)MaListe.nbElm == MaxElm);

}

//fonction d'insertion d'un entier à la fin de la liste

int insererFin(Liste \*MaListe, int Elm)

{

//teste d'existence

if (!MaListe)

{

printf("\nErreur d'allocation de la mémoire ");

return ((int)-1);

}

//tester si la liste est saturée

if (EstSaturee(\*MaListe))

{

printf("\nTable saturée, vous ne pouvez pas ajouter %d", Elm);

return ((int)0);

}

//si la liste existe et n'est pas saturée

MaListe->TabInt[MaListe->nbElm] = Elm; //insère l'élément à la fin de la liste

MaListe->nbElm++; //incrémentation de la taille de la liste

return ((int)1);

}

//fonction d'insertion d'un élément à une position donnée

int insererPos(Liste \*MaListe, int Elm, int pos)

{

int i;

//teste d'existence

if (!MaListe)

{

printf("\nErreur d'allocation de la mémoire ");

return ((int)-2);

}

//tester si la liste est saturée

if (EstSaturee(\*MaListe))

{

printf("\nTable saturée, vous ne pouvez pas ajouter %d", Elm);

return ((int)-1);

}

//tester si la position est une position valide ou non

if ((pos < 0) || (pos > MaListe->nbElm))

{

printf("\nPosition non valide");

exit((int)0);

}

//si la position est valide

for (i = MaListe->nbElm - 1; i >= pos; i--) {

MaListe->TabInt[i + 1] = MaListe->TabInt[i]; //décalage des éléments

}

MaListe->TabInt[pos] = Elm; //insère l'élément après le décalage

MaListe->nbElm++; //incrémentation du nombre d'éléments de la liste

return ((int)1);

}

//fonction de suppression dans une position donnée

int supprimerPos(Liste \*MaListe, int pos)

{

int i;

// test d'existence

if (!MaListe)

{

printf("\nErreur d'allocation de la mémoire ");

return ((int)-2);

}

// //tester si la liste est vide

if (EstVide(\*MaListe))

{

printf("\nLa liste est vide");

return ((int)-1);

}

//tester si la position est une position valide ou non

if ((pos < 0) || (pos >= MaListe->nbElm))

{

printf("\nPosition non valide");

return ((int)0);

}

//décalage des éléments

for (i = pos; i < MaListe->nbElm - 1; i++) MaListe->TabInt[i] = MaListe->TabInt[i + 1];

MaListe->nbElm--; //décrémentation du nombre d'éléments de la liste

return ((int)1);

}

//fonction qui supprime toute les occurances d'u element dans le tableau

int supprimer\_T\_occ(Liste \*MaListe,int Elm)

{

int i=0;

// test d'existance

if(!MaListe)

{

printf("\nerror d'allocation de la memoire ");

return ((int)-1);

}

// //tester si le tableau est vide

if(EstVide(\*MaListe))

{

printf("\nle tableau est vide");

return ((int)0);

}

//suppression de toutes les occurances

while(i<MaListe->nbElm)

{

//tester si l'element égale à MaTab->TabInt[i]

if(MaListe->TabInt[i]==Elm)

{

supprimerPos(MaListe,i);

i--;//si on a deux elements contigues qui sont les égaux

}

i++;//sinon incrémenter i

}

return ((int) 1);

}

//fonction d'affichage des elements du tableau

void afficher(Liste MaListe)

{

int i;

for(i=0;i<MaListe.nbElm;i++){

printf("\nTableau[%d] = %3d\n",i+1,MaListe.TabInt[i]);

}

}

//fonction qui retourne le nombre d'element de la liste

int nbrElm(Liste MaListe){

return((int)MaListe.nbElm);

}

//fonction qui supprime la 1ere occurance d'un element donnée

int supprimer\_1ere\_occ(Liste \*MaListe,int Elm){

int i=0;

// test d'existance

if(!MaListe)

{

printf("\nerror d'allocation de la memoire ");

return ((int)0);

}

//suppression de la 1ere occurance

while(i<MaListe->nbElm)

{

if(MaListe->TabInt[i]==Elm){

supprimerPos(MaListe,i);

break;

}

i++;

}

return ((int)1);

}

int main()

{

Liste \*MaListe = Creer\_Liste(); // Création de la liste

// Initialisation de la liste

if (initListe(MaListe) == 0)

{

printf("\nErreur lors de l'initialisation de la liste");

return 1;

}

printf("\nListe initialisee avec succes!");

// Tester si la liste est vide

if (EstVide(\*MaListe))

{

printf("\nLa liste est vide.");

}

// Tester si la liste est saturée

if (EstSaturee(\*MaListe))

{

printf("\nLa liste est saturee.");

}

// Ajouter des éléments à la fin de la liste

insererFin(MaListe, 10);

insererFin(MaListe, 20);

insererFin(MaListe, 30);

insererFin(MaListe, 10);

insererFin(MaListe, 30);

insererFin(MaListe, 30);

insererFin(MaListe, 20);

// Afficher les éléments de la liste

printf("\nListe apres insertion a la fin :");

afficher(\*MaListe);

// Ajouter un élément à une position donnée

insererPos(MaListe, 11, 1);

// Afficher les éléments de la liste après l'insertion

printf("\nListe apres insertion a la position 1 :");

afficher(\*MaListe);

// Supprimer un élément à une position donnée

supprimerPos(MaListe, 2);

// Afficher les éléments de la liste après la suppression

printf("\nListe apres suppression a la position 2 :");

afficher(\*MaListe);

// Supprimer toutes les occurrences d'un élément

supprimer\_T\_occ(MaListe, 10);

// Afficher les éléments de la liste après la suppression des occurrences

printf("\nListe apres suppression de toutes les occurrences de 10 :");

afficher(\*MaListe);

// Supprimer la première occurrence d'un élément

supprimer\_1ere\_occ(MaListe, 20);

// Afficher les éléments de la liste après la suppression de la première occurrence

printf("\nListe apres suppression de la premiere occurrence de 20 :");

afficher(\*MaListe);

// Afficher le nombre d'éléments de la liste

printf("\nNombre d'elements dans la liste : %d", nbrElm(\*MaListe));

// Libérer la mémoire

free(MaListe);

return 0;

}