Table des matières

1 Introduction

Le but de ce TP est de réaliser une implémentation des structures de données qui sont les listes, les piles et les files en language C à l'aide des pointeurs.

Pour chaque structure de données on devra être capable d'insérer, ou de supprimer un élément, ou encore d'afficher, copier ou savoir la taille de la structure.

2 Les listes simplement chainées

2.1 Structure de données

```
typedef struct Liste
{
  struct Liste *suiv;
  int val;
} Liste;
```

Cette structure reprèsente un élément de la liste chainée et contient deux variables :

- **suiv** : L'élément suivant cet élément.
- **val** : La valeur contenu dans cet élément.

2.2 Fonctions

2.2.1 Insértion en tête

Nom de la fonction : liste_ajout_debut

Entrées :

- Liste **liste : La liste dont laquelle on va insérer
- int val : La valeur à insérer

Description :

Insére l'élément au début de la liste, en remplacant la tête de la liste par le nouvel élément.

Code :

```
void liste_ajout_debut(Liste **liste, int val)
{
  Liste *nouveau = liste_creer(val);
  if(*liste)
    {
      Liste *tete = *liste;
      nouveau->suiv = tete;
    }
```

```
  *liste = nouveau;
}
```

2.2.2 Insértion en queue

Nom de la fonction : liste_ajout_fin

Entrées :

— Liste **liste : La liste dont laquelle on va insérer
— int val : La valeur à insérer

Description :

Insére l'élément au début de la liste, en parcourant la liste jusqu'à la fin et en attachant le nouveal élément à la queue de la liste.

Code :

```
void liste_ajout_fin(Liste **liste, int val)
{
  Liste *courant = *liste;
  Liste *nouveau = liste_creer(val);
  if(!(*liste)) *liste = nouveau;
  else
    {
      for(; courant->suiv; courant = courant->suiv);
      courant->suiv = nouveau;
    }
}
```

2.2.3 Insértion dans la liste

Nom de la fonction : liste_inserer

Entrées :

— Liste **liste : La liste dont laquelle on va insérer
— int val : La valeur à insérer
— int pos : La position de l'insertion

Description :

Insert un élément dans la position donnée, en parcourant la liste et incrémentant un compteur qui permet de savoir si on est arrivé à la bonne position.

Code :

```
void liste_inserer(Liste **liste, int val, int pos)
{
  // Conditions
  assert(pos >= 0);

  // Nouvel élément pour la valeur
  Liste *nouveau = liste_creer(val);
```

```
  // Insértion
  if(pos == 0)
    {
      liste_ajout_debut(liste, val);
    }
  else if(!(*liste))
    {
      assert(0); // Erreur  : liste vide pos doit etre 0
    }
  else
    {
      Liste *courant = *liste;
      int i;
      for(i = 0; i < pos - 1 && courant->suiv; ++i)
        courant = courant->suiv;
      if(i  != pos - 1) assert(0); // pos > taille
      nouveau->suiv = courant->suiv;
      courant->suiv = nouveau;
    }
}
```

2.2.4 Suppression d'un élement de la liste

Nom de la fonction : liste_supprimer

Entrées :

— Liste **liste : La liste de laquelle on va supprimer.
— int pos : La position de la suppression.

Description :

Supprime l'élément qui se trouve à la position indiquée, en parcourant la liste jusqu'à cette position, et en détachant l'élément de la liste.

Code :

```
void liste_supprimer(Liste **liste, int pos)
{
  assert(pos >= 0);
  assert(*liste);

  Liste *elem;
  if(!pos)
    {
      elem = *liste;
      *liste = (*liste)->suiv;
    }
  else
    {
      int i;
```

```
    Liste *courant = *liste;
    for (i = 0; i < pos - 1 && courant->suiv; ++i)
      courant = courant->suiv;
    if(i != pos - 1) assert(0); // pos > taille - 1
    elem = courant->suiv;
    courant->suiv = courant->suiv->suiv;
   }

  // Suppression de la mémoire allouée
  free(elem);
}
```

2.2.5 Copie de la liste

Nom de la fonction : liste_copier

Entrées :

— Liste **liste : Une liste

Sortie : Une copie de la liste.

Description :

Parcours la liste en ajoutant ses éléments dans une seconde liste, et retourne cette dernière.

Code :

```
Liste* liste_copier(Liste *liste)
{
  Liste *ptr = liste, *copie = NULL, *ptrCopie = NULL;

  for(; ptr; ptr = ptr->suiv) {
    if(!copie)
      {
        ptrCopie = copie = liste_creer(ptr->val);
      }
    else
      {
        ptrCopie->suiv = liste_creer(ptr->val);
        ptrCopie = ptrCopie->suiv;
      }
   }

  return copie;
}
```

2.2.6 Affichage de la liste

Nom de la fonction : liste_afficher

Entrées :

— Liste *liste : Une liste

Description :

Affiche la liste.

Code :

```
void liste_afficher(Liste *liste, Liste *fin)
{
  Liste *courant = liste;
  printf("[");
  while (courant && courant != fin)
    {
      printf("%d",courant->val);
      courant = courant->suiv;
      if(courant && courant != fin) printf(", ");
    }
  printf("]");
}
```

3 Les piles

3.1 Structure de données

La même que les listes chainées.

```
typedef Liste Pile;
```

3.2 Fonctions

3.2.1 Empilement

Nom de la fonttion : pile_empiler

Utilise la fonction d'ajout en tête d'une liste chainée.

```
#define pile_empiler liste_ajout_debut
```

3.2.2 Dépilement

Nom de la fonction : pile_depiler

Entrées :

— Pile **pile : Une pile non vide

Sortie : La valeur supprimée.

Description :

Supprime l'élément en tête de pile (en utiliant la fonction de suppression en tête de liste) et renvoie ça valeur.

Code :

```
int pile_depiler(Pile **pile)
{
  assert(*pile);

  int valeur = (*pile)->val;
  liste_supprimer(pile, 0);
  return valeur;
}
```

3.2.3 Copie de la pile

Nom de la fonction : pile_copier

Entrées :

— Pile **pile : Une pile

Sortie : Pile* : Copie de la pile.

Description :

Dépile tous les élements de la pile et les empile dans deux autres piles. L'une sera retournée et on change le pointeur "pile" pour pointer sur l'autre.

Code :

```
Pile* pile_copier(Pile **pile)
{
  Pile *copie = NULL, *pile2 = NULL;

  while(*pile)
    {
      int val = pile_depiler(pile);
      pile_empiler(&pile2, val);
      pile_empiler(&copie, val);
    }

  *pile = pile2;

  return copie;
}
```

3.2.4 Affichage de la pile

Nom de la fonction : pile_afficher

Entrées :

— Pile **pile : Une pile

Description :

Dépile et affiche les éléments de la pile un par un dans une autre pile. Puis change le pointeur “pile” pour qu'il pointe sur cette dernière.

Code :

```
void pile_afficher(Pile **pile)
{
  Pile *copie = NULL;
  while(*pile)
    pile_empiler(&copie, pile_depiler(pile));

  int i = 0;
  while(copie)
    {
      int val = pile_depiler(&copie);
      printf("%d : %d\n", i++, val);
      pile_empiler(pile, val);
    }
}
```

4 Les files

4.1 Structure de données

```
typedef Liste FileElem;

typedef struct File
{
  FileElem *tete, *queue;
}File;
```

Un élement de la file à la même structure que selui de la liste, avec ses deux variables **suiv** et **val**.

On déclare un structure **File** qui reprèsente la file et contient ça tête et ça queue.

4.2 Fonctions

4.2.1 Enfilement

Nom de la fonction : file_enfiler

Entrées :

— File *file : La file

— int val : La valeur à enfiler

Description :

Ajoute un élèment à la queue de la file. Si la pile est vide La tête et la queue vont pointer sur la même élément.

Code :

```
void file_enfiler(File *file, int val)
{
  FileElem *nouveau = calloc(1, sizeof(FileElem));
  nouveau->val = val;


  if(!file->tete)
    {
      file->tete = file->queue = nouveau;
    }
  else
    {
      file->queue->suiv = nouveau;
      file->queue = file->queue->suiv;
    }
}
```

4.2.2 Défilement

Nom de la fonction : file_defiler

Entrées :

— File *file : Une file non vide

Sortie : int : La valeur supprimée.

Description :

Supprime l'élément en tête de la file.

Code :

```


int file_defiler(File *file)
{
  assert(file->tete);
  assert(file->queue);

  int valeur = file->tete->val;


  if(file->tete == file->queue)
    {
      free(file->queue);
```

```
    file->tete = file->queue = NULL;
   }
  else
   {
    FileElem *suppr = file->tete;
    file->tete = file->tete->suiv;
    free(suppr);
   }

  return valeur;
}
```

4.2.3 Copie de la file

Nom de la fonction : file_copier

Entrées :

— File *file : Une file

Sortie : File* : Une copie de la file

Description :

Défile les éléments de la file un par un et les enfile dans deux autres files. L'une sera retournée comme copie et on change le pointeur "file" pour pointer sur l'autre.

Code :

```
File file_copier(File *file)
{
  File copie = {NULL, NULL}, file2 = {NULL, NULL};

  while(file->tete)
   {
    int val = file_defiler(file);
    file_enfiler(&file2, val);
    file_enfiler(&copie, val);
   }

  *file = file2;
  return copie;
}
```

4.2.4 Affichage de la file

Nom de la fonction : file_afficher

Entrées :

— File *file : Une file

Description :

Défile les éléments de la file un par un et les affiche. Le pointeur “file” et alors changé pour pointer sur la nouvel file.

Code :

```
void file_afficher(File *file)
{
  File copie = {NULL, NULL};

  printf("[");
  while(file->tete)
    {
      int valeur = file_defiler(file);
      printf("%d", valeur);
      if(file->queue) printf(", ");
      file_enfiler(&copie, valeur);
    }

  printf("]");

  *file = copie;
}
```