Programmation et Structure de Données

Structure de données de Base
Les listes chainées Partie 3

ENS KATEB

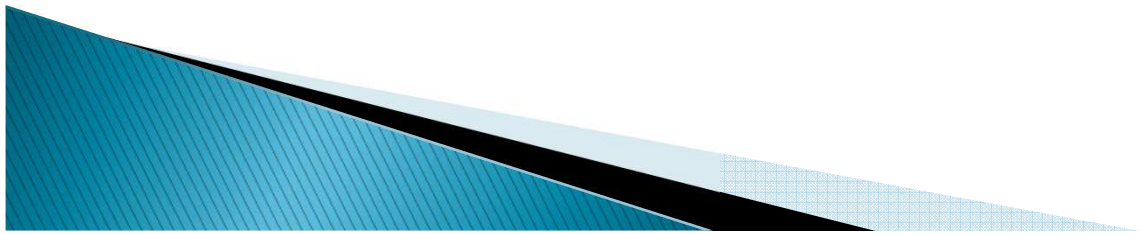
Suppression d'un élément d'une liste

Dans une liste on a trois cas de suppressions :
Suppression de la tête (début) de liste, Suppression du dernier élément : en queue de Liste ou Suppression d'un élément ayant une position précise.

Cas d'une liste simple :

Supprimer la tête de liste :

```
elem_s * supprD_s (elem_s *L)
{
  if(L==NULL) cout  <<"\nImpossible de supprimer sur une liste Vide!!!";
  else
      {
            elem_s * E = L;
            L = L->suiv ;
            delete(E);
            cout  <<"\nSuppression effectuee avec success!!";
      }
  return L;
}
```



Suppression d'un élément d'une liste

Cas d'une liste simple :
Supprimer la Queue de liste :

```
elem_s * supprF_s (elem_s *L)
{
  if(L==NULL) cout  <<"\nImpossible de supprimer sur une liste Vide!!!";
  else
     {
            if(L->suiv==NULL){delete(L);L=NULL;}
            else
               {
                  elem_s *T = L;
                  while (T->suiv->suiv != NULL) T = T->suiv;
                  delete(T->suiv);
                  T->suiv = NULL;
               }
             cout  <<"\nSuppression effectuee avec success!!";
      }
  return L;
}
```

Suppression d'un élément d'une liste

Cas d'une liste simple :
Supprimer un élément connu (soit son emplacement (position) soit son contenu):

Ex : supprimer un élément contenant une valeur = val :

```
elem_s * supprM_s (elem_s *L, int val)
{//1
  if(L==NULL)
     cout  <<"\nImpossible de supprimer sur
                  une liste Vide!!!";
  else
     {//2
       elem_s *T;
       if(L->D==val)
        {//3
         T = L;   L=L->suiv;    delete(T);
         cout  <<"\nSuppression effectuee
                      avec success!!";
         } //3
  else
     {//4
       T = L;  elem_s *M = L->suiv;
       while (M != NULL && M->D !=val)
            {T = T->suiv;M = M->suiv;}
       if (M != NULL)
        {//5
          T->suiv = M->suiv;  delete(M);
          cout  <<"\nSuppression effectuee avec
                       success!!";
        } //5
     } //4
   } //2
  return L;
} //1
```

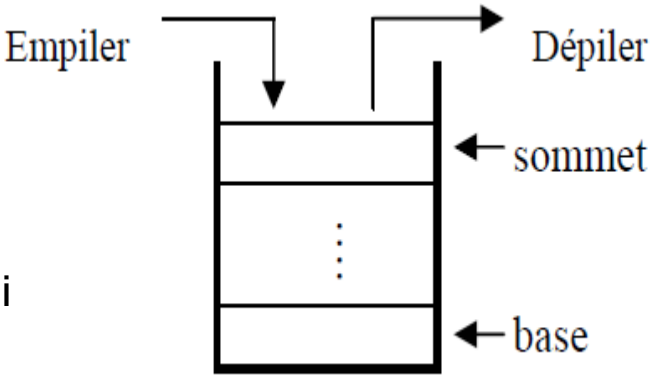
Cas particuliers de listes : Piles

Une pile est une liste chaînée dans laquelle :
- Un élément ne peut être ajouté qu'au sommet (tête) de la pile,
- Un élément ne peut être retiré que du sommet (tête) de la pile.

Il s'agit donc d'une structure de type LIFO (Last In First Out). On ne travaille que sur le sommet de la pile.

Opérations:
- PUSH (empiler) : ajouter un élément,
- POP (dépiler) : supprimer un élément.
- Is_empty (est_vide) : vérifier si la pile est vide

Exercice : écrire une fonction qui retourne true si une pile est vide et false sinon

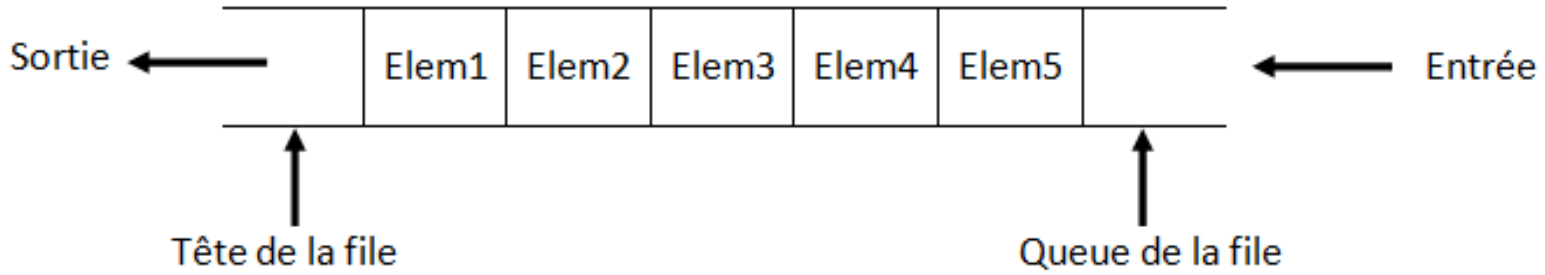


Cas particuliers de listes : Files

Une file, ou file d'attente, est une liste chaînée dans laquelle :
- Un élément ne peut être ajouté qu'à la queue de la file,
- Un élément ne peut être retiré qu'à la tête de la file.

Il s'agit donc d'une structure de type FIFO (First In First Out). Les données sont retirées dans l'ordre où elles ont été ajoutées.

Opérations:
- Enfiler : toujours à la queue et jusqu'à la limite de la mémoire,
- Défiler : toujours à la tête si la file n'est pas vide,
- Est_vide : vérifier si la file est vide ou non.



Exemple (Piles):

Supposant une pile d'entiers qui est triée par ordre croissant (voir la figure)

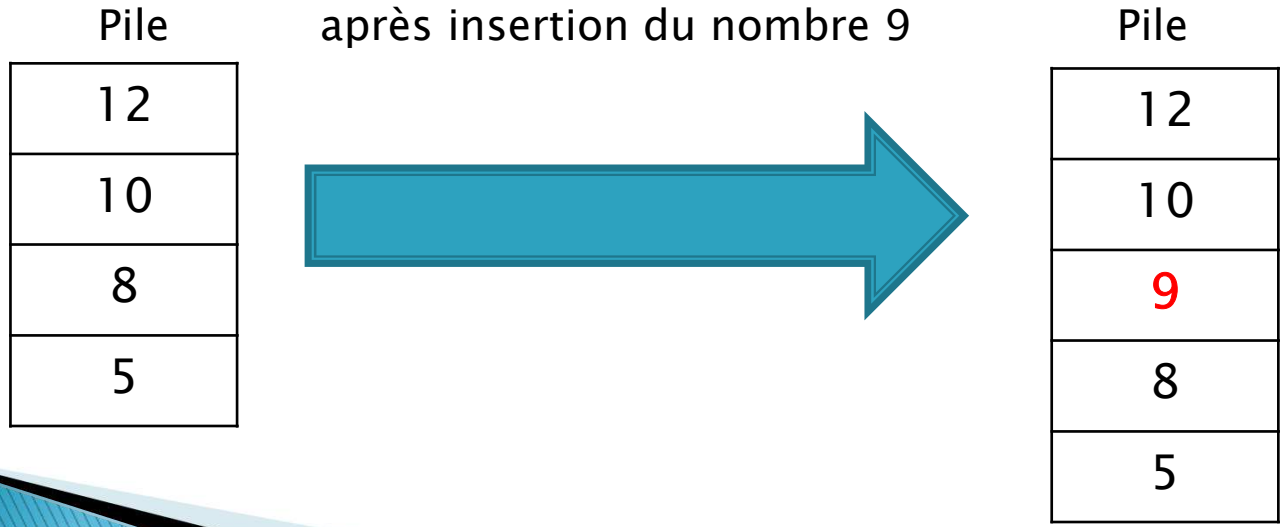
1 – Ecrire les sous programmes suivants :

a – elem* Empiler (elem *L, int X) où L représente la plie et X représente la valeur de l'élément à empiler,

b – elem* Depiler (elem *L, int &X) où L représente la plie et X représente un argument où on stocke la valeur de l'élément à dépiler

c – bool Vide (elem *L) retoune vrais si la pile est vide et faux sinon.

2 – Ecrire un sous programme qui prend en argument une pile triée par ordre croissant et un nombre entier, puis insère ce nombre dans la pile, mais à condition que la pile restera triée après l'insertion



Exemple (Piles): Solution

```
elem* Empiler (elem *L, int X)
{
  elem * e = new (elem);
  e->D = X;  e->suiv = L;
  L = e;
  return L;
}
```

```
elem* Depiler (elem *L, int &X)
{
 if(!Vide(L)) // ou L!= NULL
  {
   elem * e = L;    L = L->suiv;
       X = e->D;    delete (e);
    }
  return L;
}
```

```
bool Vide (elem * L)
{
  if(L==NULL)
     return true;
  else
     return false;
}
```

```
elem* Ajouter (elem *L, int A)
{
  elem *T = NULL;     int B;
  if(L!=NULL)
    while (!Vide(L) && L->D>A)
      {
        Depiler (L, B):
        Empiler (T, B);
      }
  Empiler (L, A) ;
  // suite
  while (!Vide(T))
    {
      Depiler (P, B);
      Empiler (L, B);
    }
  return L;
}
```

Exemple (Files):

Supposant une file d'entiers

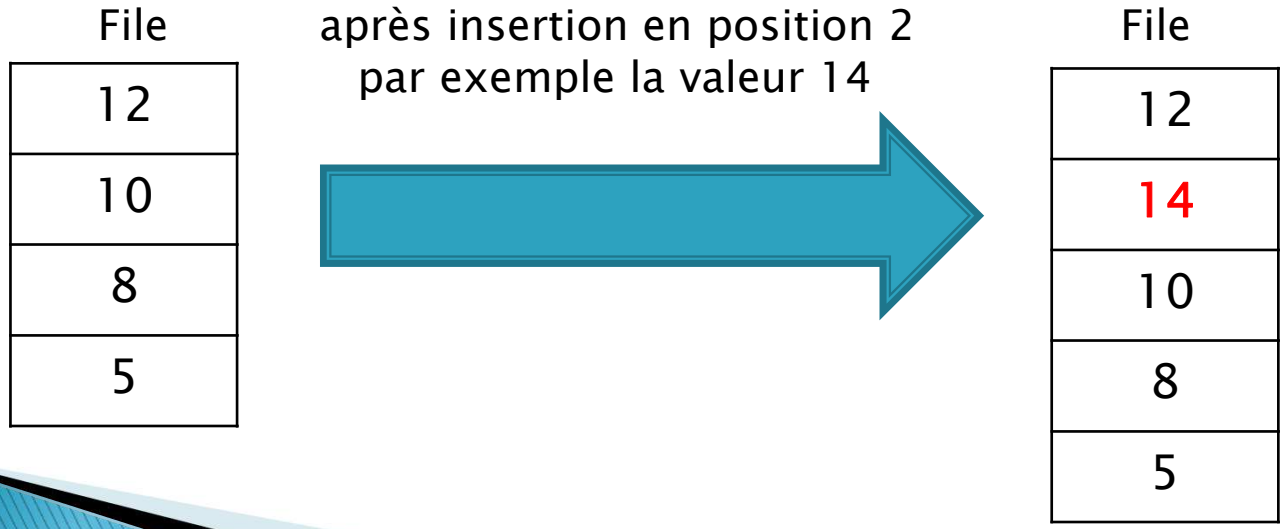
1 – Ecrire les sous programmes suivants :

a – elem* Enfiler (elem *L, int X) où L représente la flie et X représente la valeur de l'élément à enfiler,

b – elem* Defiler (elem *L, int &X) où L représente la flie et X représente un argument où on stocke la valeur de l'élément à défiler

c – bool Vide (elem *L) retoune vrais si la file est vide et faux sinon.

2 – Ecrire un sous programme qui prend en argument une file et un nombre entier pos, puis insère un élement entré par l'utilisateur à la position pos dans la file.



Exemple (Files): Solution

```
elem* Enfiler (elem *L, int X)
{
  elem * e = new (elem);
  e->D = X;  e->suiv = NULL;
  if (L==NULL) L = e;
  else
   {
    elem *T = L;
    while (T->suiv!=NULL)
        T= T->suiv;
    T->suiv = e;
   }
  return L;
}
```

```
elem* Defiler (elem *L, int &X)
{
 if(!Vide(L)) // ou L!= NULL
  {
   elem * e = L;    L = L->suiv;
       X = e->D;    delete (e);
  }
  return L;
}
```

```
bool Vide (elem * L)
{
  if(L==NULL)
     return true;
  else
     return false;
}
```

```
elem* Ajouter (elem *L, int pos)
{
  elem *T = NULL;     int i,B,A;
  if(pos>=1 && pos <=NBR(L)+1)
    {
      for (i=0; i<pos-1; i++)
        {
          Defiler (L, B);
          Enfiler (T, B);
        }
      cout <<''Entrer une valeur : ''; cin>>A;
/* NBR : fonction qui retourne le nombre des
   éléments de la file */
    // suite
    Enfiler (T, A);
    while (!Vide(L))
     {
       Defiler (L, B);
       Enfiler (T, B);
     }
    return T;
}
else
 {
   cout <<''Erreur de position !!'';
   return L;
 }
}
```