Rappel

Listes chainées, Piles, Files

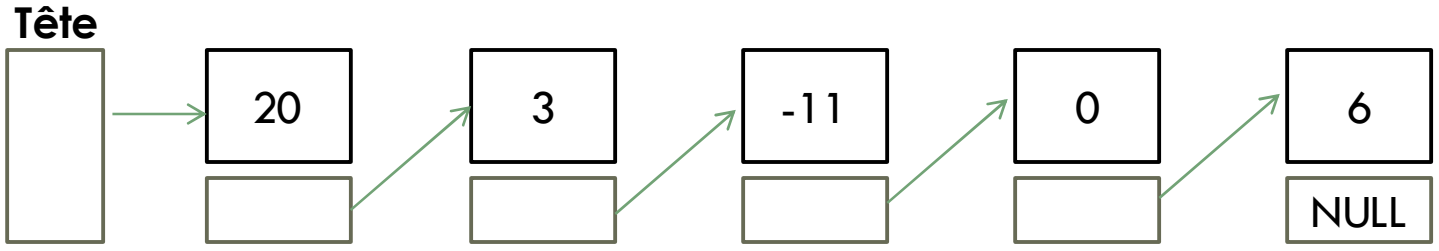
- Lorsqu'une structure contient une donnée avec un pointeur vers un élément de même composition, on parle alors de liste chainée.
  - Les listes chainées sont basées sur les pointeurs et sur les structures;
  - Quand une variable pointeur ne pointe sur aucun emplacement, elle doit contenir la valeur ***Nil*** - Not In List (qui est une adresse négative).

Par définition, une liste chaînée est une structure linéaire qui n'a pas de dimension fixée lors de sa création.

- Ses éléments de même type sont éparpillés dans la mémoire et reliés entre eux par des pointeurs;
- Chaque élément (dit ***nœud***) est lié à son successeur. Chaque prédécesseur contient le pointeur du successeur;
- Le dernier élément de la liste ne pointe sur rien (***Nil***);
- La liste est uniquement accessible via sa tête de liste qui est son premier élément.

Rappel

Listes chainées, Piles, Files



- Tête est le pointeur contenant l'adresse du premier élément alors que chaque nœud est une structure avec une case contenant la valeur à manipuler (20, 3, -11, 0 et 6) et une case contenant l'adresse de l'élément suivant;
- Contrairement au tableau, les éléments n'ont aucune raison d'être voisins ni ordonnés en mémoire;
- Selon la mémoire disponible, il est possible de rallonger ou de raccourcir une liste;
- Pour accéder à un élément de la liste il faut toujours débuter la lecture de la liste par son premier élément dans le pointeur duquel est indiqué la position du deuxième élément. Dans le pointeur du deuxième élément de la liste on trouve la position du troisième élément. Ainsi de suite jusqu'à obtenir la position de l'élément… ;
- Pour ajouter, supprimer ou déplacer un élément il suffit d'allouer une place en mémoire et de mettre à jour les pointeurs des éléments.

Rappel

Listes chainées, Piles, Files

Il existe différents types de listes chaînées :

- *Liste chaînée simple* constituée d'éléments reliés entre eux par des pointeurs;
- *Liste doublement chaînée* où chaque élément dispose de deux pointeurs pointant respectivement sur l'élément précédent et l'élément suivant. *Ceci permet donc la lecture dans les deux sens;*
- *Liste circulaire* où le dernier élément pointe sur le premier élément de la liste.

Rappel

Listes chainées, Piles, Files

exemples: insertion/suppression par l'avant

```
#include<iostream>
using namespace std;

int main(){
        int pos_noeud, num_noeud;
        typedef struct noeud
        {
                int data;       // pour stocker l'information
                noeud *suivant; // reference au noeud suivant
        };
        // insertion par l'avant
        noeud *tete = NULL;
        // premier noeud
        noeud *noeud1 = new noeud;
        noeud1->data=10;
        noeud1->suivant=tete;
        tete = noeud1;
        // deuxiéme noeud
         noeud *noeud2 = new noeud;
        noeud2->data=20;
        noeud2->suivant=tete;
        tete = noeud2;
        // Affichage
        cout<<"TETE -> ";
         while(tete!=NULL)
        {
                cout<< tete->data <<" -> ";
                 tete = tete->suivant;
        }
        cout<<"NULL";

         return 0;
}
```

```
// suppression par l'avant
noeud *cellule=new  noeud;
cellule=tete;
tete=cellule->suivant;
delete cellule;
```

Rappel

Listes chainées, Piles, Files

exemples: Insertion à une position spécifique

```
// ……
cout<<"Entrer la position du noeud: ";
cin>>pos_noeud;
noeud *curseur=new noeud;
curseur->suivant=tete;
for(int i=1;i<pos_noeud;i++){
    curseur=curseur->suivant;
    if(curseur==NULL){
        cout<<"La position"<<pos_noeud<<" n'est pas dans la liste"<< endl;
        break;
    }
}
noeud *nouveau=new noeud;
nouveau->data=30;
nouveau->suivant=curseur->suivant;
curseur->suivant=nouveau;
// …
```

Rappel

Listes chainées, Piles, Files

exemples: insertion/suppression par l'arrière

```
#include<iostream>
using namespace std;

int main(){
        typedef struct noeud
        {
                int data;       // pour stocker l'information
                noeud *suivant; // reference au noeud suivant
        };
        // insertion par l'arrière
        noeud *tete = NULL;
        // premier noeud
        noeud *noeud1 = new noeud;;
        tete = noeud1;
        noeud1->data=10;
        noeud1->suivant=NULL;
        // deuxiéme noeud
        noeud *noeud2 = new noeud;
        noeud1->suivant = noeud2
        noeud2->data=20;
        noeud2->suivant=NULL;
        // Affichage
        cout<<"TETE -> ";
         while(tete!=NULL)
        {
                cout<< tete->data <<" -> ";
                 tete = tete->suivant;
        }
        cout<<"NULL";

         return 0;
}
```

```
// suppression par  l'arrière
noeud *cellule=new  noeud;
cellule=tete;
noeud *encien=new  noeud;
while(cellule->suivant!=NULL)
{
        encien=cellule;
        cellule=cellule->suivant;
}
encien->suivant=NULL;
delete cellule;
```

Rappel

Listes chainées, **Piles, Files**

Les ***piles*** et les ***files*** sont des *listes chaînées particulières* permettant d'ajouter et de supprimer des éléments uniquement à une des deux extrémités de la liste.

- Une structure **pile** est assimilable à une superposition d'assiettes . on pose et on prend à partir du sommet de la pile. C'est du principe **LIFO** (Last In First Out);
- Une structure **file** est assimilable à une file d'attente de caisse. le premier client entré dans la file est le premier à y sortir. C'est du principe **FIFO** (First In First Out).

Rappel

Listes chainées, **Piles,** Files

une ***Pile*** est donc un ensemble de valeurs ne permettant des insertions ou des suppressions qu'a une seule extrémité, le ***sommet***.

- l'opération insertion d'un objet sur une pile consiste à empiler cet objet au sommet de celle-ci. **Exemple**: ajouter une nouvelle assiette au dessus de celle qui se trouve au sommet.
- l'opération suppression d'un objet sur une pile consiste à dépiler celui-ci au sommet de celle-ci. **Exemple**: supprimer ou retirer l'assiette qui se trouve au sommet.

Une pile sert essentiellement à stocker des données ne pouvant pas être traitées immédiatement.

Rappel

Listes chainées, **Piles,** Files

une ***Pile*** est donc un ensemble de valeurs ne permettant des insertions ou des suppressions qu'a une seule extrémité, le ***sommet***.

- l'opération insertion d'un objet sur une pile consiste à empiler cet objet au sommet de celle-ci. **Exemple**: ajouter une nouvelle assiette au dessus de celle qui se trouve au sommet.
- l'opération suppression d'un objet sur une pile consiste à dépiler celui-ci au sommet de celle-ci. **Exemple**: supprimer ou retirer l'assiette qui se trouve au sommet.

Une pile sert essentiellement à stocker des données ne pouvant pas être traitées immédiatement.

Rappel

Listes chainées, **Piles,** Files

une ***Pile*** est un enregistrement avec une variable sommet indiquant le sommet de la pile et une structure données pouvant enregistrer les données.

La manipulation d'une pile en C++ nécessite d'inclure la bibliothèque stack. Dans cette bibliothèque nous trouvons les fonctions pour:

- La déclaration. syntaxe: stack<type> pile;
- Connaitre la taille de la pile (qui nous renvoie le nombre d'élément): pile.size();
- Vérifier si la pile est vide ou non: pile.empty();
- Ajouter une nouvelle valeur à la pile(empiler): pile.push(element);
- Accéder au premier élement de la pile: pile.top();
- Supprimer la valeur se trouvant au sommet de la pile(depiler): pile.pop(); // ici, la pile ne doit pas être vide !

Rappel

Listes chainées, **Piles,** Files

exemple

```
#include<iostream>
#include<stack>
using namespace std;
int main(){
		int n;
		stack<int> pile;
		// remplissage
		cout << "veuillez saisir un element: ";
		cin >> n;
		while(n>0){
				pile.push(n);
				cout << "entrer un autre element: ";
				cin >> n;
		}
		// affichage
		cout << endl;
		if(pile.size()==0){
				cout <<"la pile est vide ";
		}else if(pile.size()==1){
				cout<<"la pile contient un element qui est: "<<pile.top();
		}else{
				cout <<"la pile contient " << pile.size() << " elements que sont :" << endl;
				while(!pile.empty()){
						cout << pile.top() << " ";
						pile.pop();
				}
		}

		return 0;
}
```

Rappel

Listes chainées, **Piles,** Files

Pile et fonction

Passer une pile en paramètre à un sous-programme se fait par références.

Exemple: remplissage(stack<int>& pile) , affichage(stack<int>& pile);

```
...
void remplissage(stack<int>& pile)
{
		cout << "veuillez saisir un element: ";
		cin >> n;
		while(n>0){
				pile.push(n);
				cout << "entrer un autre element: ";
				cin >> n;
		}
}
------
void affichage(stack<int>& pile)
{
		cout << endl;
		if(pile.size()==0){
				cout <<"la pile est vide ";
		}else if(pile.size()==1){
				cout<<"la pile contient un element qui est: "<<pile.top();
		}else{
				cout <<"la pile contient " << pile.size() << " elements que sont :" << endl;
				while(!pile.empty()){
						cout << pile.top() << " ";
						pile.pop();
				}
		}
}
```

Rappel

Listes chainées, Piles, **Files**

une ***File*** est donc un enregistrement avec une variable ***Début*** indiquant le premier élément, ***Queue*** indiquant le dernier élément et une structure données pouvant enregistrer les données.

La manipulation d'une **file** en **C++** nécessite d'inclure la bibliothèque queue. Dans cette bibliothèque nous trouvons les fonctions pour:

- La déclaration. syntaxe: queue<type> file;
- Connaitre la taille de la file (qui nous renvoie le nombre d'élément): file.size();
- Vérifier si la file est vide ou non: file.empty();
- Ajouter une nouvelle valeur à la pile(empiler): file.push(element);
- Accéder au premier élément de la file: file.front();
- Accéder au dernier élément de la file: file.back();
- Supprimer le premier élément de la file(depiler): file.pop(); // ici, la file ne doit pas être vide !

Rappel

Listes chainées, Piles, **Files**

```
#include<iostream>
#include<queue>
using namespace std;
int main(){
		int n;
		queue<int> file;
		// remplissage
		cout << "veuillez saisir un element: ";
		cin >> n;
		while(n>0){
				file.push(n);
				cout << "entrer un autre element: ";
				cin >> n;
		}
		// affichage
		if(file.size()==0){
				cout <<"la file est vide ";
		}else if(file.size()==1){
				cout<<"la file contient un element qui est: "<<file. front();
		}else{
				cout <<"la file contient " << file.size() << endl;
				cout << "Le premier élément est : " << file.front() << endl;
				cout << "Le dernier élément est : " << file.back() << endl;
				cout <<  "Les elements sont :" << endl;
				while(!file.empty()){
						cout << file.front() << " ";
						file.pop();
				}
		}

		return 0;
}
```