Analyse et Programmation Orientées Objets / C++

La classe **list** (STL)

Présentation de la classe

- **Chaînage avant et arrière**
- **Accès direct à la tête et à la queue de la liste**
- **Itérateurs bidirectionnels**

Créer et initialiser une liste

```
#include <list>
using namespace std;
void main () {
list <int>  exemple;
list <int>::iterator i= exemple.begin();

      exemple.insert(i, 17) ;
      exemple.insert(i, 14);
      exemple.insert(i, 3, 11);
      ---
```

Visualiser une liste d'entiers

```
---
void visualiser (char* titre, list<int>& cible) {
int taille= cible.size();
    cout << titre << "[";
    if (taille==0) {cout << "]"; return;}
    list<int>::iterator k= cible.begin();
    while (taille >1) {cout <<  *k++ <<",  " ; taille--;}
    cout  << *k  << "]";
}
```

Parcourir une liste de réels

```
---
float moyenne (list<float>& notes) {
list<float>::iterator k= notes.begin();
float somme=0.0f;

      while (k != notes.end()) somme += *k++;
      return somme/notes.size();
}
```

Dupliquer une liste

```
---
void main () {
list <int>  exemple, travail;
list <int>::iterator i= exemple.begin();

      exemple.insert(i, -1) ; exemple.insert(i, 11);
      exemple.insert(i, -7);
      ---
      travail= exemple;
```

Comparer deux listes entre elles

```
---
void main () {
list <int>  op1, op2;
    list<int>::iterator i= op1.begin(), k=op2.begin();
    op1.insert(i, 4) ; op1.insert(i, -5);
    op2.insert(k, -5); op2.insert(k, 4);
     ---
     cout  <<  ”Resultat : " <<  (op1==op2) << endl;
```

Insérer un nouvel élément

```
---
void main () {
list <int>  exemple;
list <int>::iterator i= exemple.begin();

    exemple.insert(i,  22);  exemple.insert(i, -27)
    exemple.insert(i,  15);  exemple.insert(i, 58) ;
     ---
     i= exemple.begin(); i++; i++;
     exemple.insert(i, -44)
```

Supprimer un élément cible

```
---
void main () {
list <int>  exemple;
list <int>::iterator i= exemple.begin();

    exemple.insert(i,  22);  exemple.insert(i, -27)
    exemple.insert(i,  15);  exemple.insert(i, 58) ;
     ---
     i= exemple.begin(); i++; i++;
     i=exemple.erase(i);
```

Supprimer une valeur cible

```
---
void main () {
list <int>  ages;
list <int>::iterator j= ages.begin();

     ages.insert(j,  18); ages.insert(j, 14)
     ages.insert(j,  0);   ages.insert(j, 22) ;
      ---
      ages.remove(0) ;
```

Supprimer les doublons consécutifs

```
---
void main () {
list <int>  cible;
list <int>::iterator i= cible.begin();

      cible.insert(i, -2) ; cible.insert(i, 7);
      cible.insert(i, -3); cible.insert(i, -1);
      cible.insert(i, -1); cible.insert(i, 7);
      ---
      cible.unique();
```

Trier une liste (ordre croissant)

```
---
void main () {
list <int>  exemple;
list<int>::iterator i= exemple.begin();

      exemple.insert(i, 17) ; exemple.insert(i, 8);
      exemple.insert(i, 11); exemple.insert(i, 5);
      ---
      exemple.sort();
```

Inverser une liste

```
---
void main () {
list <int>  mesures;
list <int>::iterator i= mesures.begin();

      mesures.insert(i, 4) ; mesures.insert(i, 5);
      mesures.insert(i, 36);
      ---
      mesures.reverse();
```

Transférer le contenu d'une liste

```
---
void main () {
list <int>  source, transfert;
list <int>::iterator i= source.begin();

      source.insert(i, -1) ; source.insert(i, 11);
      source.insert(i, -7);
      ---
      transfert.swap(source);
```

Récapitulatif partiel

- **Méthodes des conteneurs et séquences**

  begin, end, insert, size, swap, empty, erase, clear, ...

  Opérateurs : =, ==, != et sous conditions < et >

- **Méthodes propres aux listes**

  push_back, push_front, pop_front, pop_back, remove, remove_if, unique, sort, splice, merge, reverse, ...