List In STL;

작성자 : 201907010 김민철

목 차

- *자료 구조 List란*

→순차 리스트

→단순 연결 리스트

→이중 연결 리스트

- *STL에서의 List*

→ 실제 사용 예시

자료 구조 List란

#List는 **순서가 있는 데이터의 모임**으로서 **시퀀스**라는 이름으로 불리기도 한다.

#데이터가 한 줄로 저장되어 있는 **선형 구조**를 띄고 있으며 원하는 데이터에 접근하여 원하는 작업을 수행할 수 있다. 이때, 데이터에 접근 하는 방식은 리스트의 형태에 따라 달라진다.

#리스트 내부에 비어 있는 데이터를 허용하지 않기 때문에 데이터를 삭제할 때 마다 더 이상 삭제된 데이터에 접근하지 못하도록 하는 과정을 거친다.

#첫 번째 데이터를 **Head**라고 하고, 마지막 데이터를 **Tail**이라고 한다.

#형태로는 순차 리스트, 연결 리스트, 이중 연결 리스트, 원형 연결 리스트가 있다.

순차 리스트

#Array List라고도 하며, **배열을 이용**하여 리스트를 구현한 형태이다.

#인덱스를 이용해 값에 접근하여 작업을 수행할 수 있지만, 리스트는 비어 있는 데이터를 허용하지 않기 때문에 데이터를 삭제하면 삭제한 데이터보다 인덱스 번호가 뒤쪽에 있던 데이터들을 앞으로 옮겨야 한다.

#리스트 중간에 데이터를 추가하기 위해서는 추가를 원하는 인덱스 번호보다 뒤쪽에 있는 데이터들을 먼저 뒤로 옮기고 데이터가 비워진 인덱스 번호에 원하는 데이터를 삽입해야 한다.

#위와 같은 특징들로 인해 순차 리스트는 **데이터의 삭제 및 삽입 속도**는 느려지지만 인덱스 번호를 통해 접근이 가능하므로 **데이터 탐색 속도**는 빨라지게 된다. 또한 인덱스 번호가 단순히 몇 번째 데이터인지만 의미하기 때문에 **인덱스 번호를 데이터의 식별자로 사용하기에는 제한된다**.

단순 연결 리스트

#Linked List라고도 하며, **노드**들의 연결로 이루어져 있다.

#노드는 데이터와 포인터 필드 이루어져 있다. 데이터 필드에는 데이터를 저장하고, 포인터 필드에는 **다음 노드의 주소를 저장**한다.

#단순 연결 리스트에서 특정 데이터에 접근하기 위해서는 원하는 데이터를 찾을 때 까지 첫 노드부터 포인터를 통해 한 노드씩 이동하며 탐색해야 한다. 따라서 **원하는 데이터를 찾는데 걸리는 시간이 오래 걸린다.**

#리스트 내부에 노드를 삭제하거나 추가하게 되면 그 노드의 **이전 노드가 가리키는 노드의 주소를 변경**하는 방법으로 작업을 수행한다.

이중 연결 리스트

#Doubly Linked List라고도 한다.

#노드들의 연결로 구성되어 있지만, 각 노드는 자기 다음 노드의 주소 뿐만 아니라 자기 앞의 노드의 주소 또한 가리키고 있다. 따라서 단순 연결 리스트와는 달리 **Tail에서 Head 방향으로도 탐색이 가능**하게 된다.

#리스트 내부에서 노드를 삭제하거나 추가하면 그 노드의 **이전 노드와 다음 노드**가 가리키는 **주소의 값을 변경**하는 방법으로 작업을 수행한다.

#이중 연결 리스트는 **양방향 탐색이 가능**하다는 장점 때문에 **메모리를 더 많이 사용**한다는 단점에도 불구하고 단순 연결 리스트보다 높은 사용율을 보인다.

STL에서의 List

#STL의 List는 **이중 연결 리스트 구조**이다.

#list<*datatype*> *list_name*;과 같은 형태로 생성한다.

#list_name 뒤에 **(number)**를 추가하면 디폴트 값을 가지는 요소를 number개 가진 리스트가 생성된다.

#list_name 뒤에 **(number1, number2)**를 추가하면 number2 값을 가지는 요소를 number1개 가진 리스트가 생성된다.

#**동적 할당**을 기본적으로 제공한다.

#**반복자**(Iterator)로 원하는 원소에 접근한다. 반복자는 **list<*datatype*>::iterator *iterator_name*;**으로 생성할 수 있다.

STL에서의 List

#STL에서 제공하는 List 관련 함수 중 대표적인 함수들은 다음과 같다.

(아래 표에서 iter는 반복자를 의미하고, x는 요소를 의미한다.)

List.push_back(x); List.push_front(x);	리스트의 끝이나 시작에 요소 추가
List.pop_back(); List.pop_front();	리스트의 끝이나 시작에 있는 요소를 제거
List.remove(x); List.remove_if(predicate);	지정한 값이나 조건과 일치하는 요소를 제거
Iter = List.erase(iter); Iter = List.erase(iter_begin, iter_end);	지정한 위치 또는 범위에 있는 요소를 제거하고 다음 요소의 반복자를 반환
List.insert(iter, x);	지정한 위치에 요소를 추가
List.sort(); List.sort(predicate);	리스트를 오름차순 또는 조건으로 정렬
List.unique();	리스트 내부에서 인접하는 중복 요소를 제거
List.size();	리스트에 있는 요소들의 수를 반환
List.empty();	리스트가 비어 있는지 판단

실제 사용 예시 (Backjoon 1406번)

#include <list> → 리스트 자료 구조를 사용하기 위한 선언

```
#include <iostream>

using namespace std;

int M;
char c1;
string s;
```

list<char> lst; → 리스트 생성
list<char>::iterator it; → 리스트의 요소들에 접근하기 위한 반복자 생성

```
int main (){

        cin >> s;

        for (int i = 0; i < s.length(); i++)
        {
                lst.push_back(s[i]);→ 리스트에 값을 삽입
        }
```

it = lst.end(); → it에 마지막 요소의 반복자를 대입

```
        cin >> M;

        for (int i = 0; i < M; i++)
        {
                cin >> c1;
                if (c1 == 'P')
                {
                        char c2;
                        cin >> c2;
                        lst.insert(it, c2); →반복자 위치에 요소 삽입
                        }
                if (c1 == 'L')
                {
                        if (it != lst.begin())
                        {
                        --it;→반복자가 가리키는 위치값 감소
                        }
                }
                if (c1 ==  ' D')
                {
                        if (it != lst.end())
                        {
                        ++it;→반복자가 가리키는 위치값 증가
                        }
                }
                if (c1 == 'B')
                {
                        if (it != lst.begin())
                        {
                        --it;
                        it = lst.erase(it);→반복자 위치의 요소 삭제
                        }
                }
        }
```

for (it = lst.begin(); it != lst.end(); it++) → 리스트가 끝날 때 까지 반복

```
        {
                cout << *it;→ 반복자가 가리키는 곳의 요소 출력
        }
}
```

이상입니다;