4주차 결과보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181294 이름: 임승섭

**1.**

<Stack.h>

template <class T>

class Stack : public LinkedList<T>{

    public:

        bool Delete (T &element){

            //first가 0이면 false반환

            if(this->first == 0)

                return false;

            // LinkedList와 달리 Stack은 current가 가리키는 곳을 삭제

            else {

                Node<T> \*current = this->first;

                element = this->first->data;

                this->first = current->link;

                delete current;

                this->current\_size--;

                return true;

            }

        }

};

우선, 이번 실습에서는 Template을 이용한 파라미터적 다형성을 보였다. 원래 int 자료형만 사용할 수 있었지만 이를 통해 data의 자료형을 고민할 필요가 없어지게 된다. 이 파라미터적 다형성을 지원하기 위해 코드 처음에 template <class T> 를 작성함으로써 Template class로 확장하였다.

Stack은 기본적으로 LinkedList 를 상속받기 때문에 Stack class 를 선언하는 부분 직후에 public LinkedList<T> 를 작성함으로써 LinkedList를 상속받는다는 것을 보였다.

LinkedList.h 파일에는 Getsize, Insert, Delete, Print 함수가 구현되어 있다. 우리는 현재 data의 자료형을 T로 사용하기 때문에 각각의 함수 선언 부분에 element들도 T 타입으로 들어갈 것을 수정해주어야 한다.

<LinkedList.h>

template <class T> void LinkedList<T>::Insert(T element){

template <class T> bool LinkedList<T>::Delete(T &element){

template <class T> void LinkedList<T>::Print(){

위 함수들 중 Delete 함수는 Stack.h 파일에서 직접 수정이 필요하다. LinkedList 같은 경우에는 Node를 Delete할 때 queue의 형식처럼 맨 뒤의 노드를 삭제하지만, Stack은 맨 앞의 노드를 삭제해야 하기 때문이다. 따라서 Delete 함수를 Stack class 에서 public으로 선언한 후 새롭게 정의를 해 주었다.

우선 함수의 인자는 T 타입으로 선언해 주었다. 다음 줄에서는 first가 0, 즉 삭제를 할 노드가 없는 경우에는 false를 반환해 주었다. 그렇지 않은 경우에는 노드가 1개 이상이기 때문에 첫 번째 노드, 즉 first를 삭제해준다. 그러기 위해서는 first가 가리키는 link(노드)를 first로 바꿔주어야 한다. 그래야 그 이후에도 first를 계속 사용할 수 있기 때문이다.

직접적으로 first를 사용하면 다시 복구해야 하는 작업이 필요하기 때문에 current 라는 Node를 생성해서 first를 나타내게 하였다. element 는 first의 data로 선언하였고, first를 current의 link로 바꿔주었다. 필요한 작업이 끝난 이후에 current를 소멸자 delete를 사용하여 삭제하였고 전체 노드의 개수를 줄였기 때문에 current\_size--를 해주었다. 성공적으로 노드를 삭제했기 때문에 true를 반환해준다.

<main.cpp>

                case 2:

                    if(p->Delete(tmpItem)==true)

                        cout<<tmpItem<<"가 삭제되었습니다."<<endl;

                    else cout<<"비어있습니다. 삭제 실패"<<endl;

                    break;

만약 Delete함수가 성공적으로 삭제를 마치고 true를 반환한다면 화면에는 그 element가 삭제되었다고 출력한다. 만약 false를 반환하면 삭제 실패했다는 문장을 출력하고 반복문이 종료된다.

실습 중 오류가 났던 부분은 Delete함수를 실행할 때 한번 삭제했던 element가 그 다음에도 계속 함수를 실행할 때마다 삭제된다고 출력되는 부분이었다. Delete 함수를 선언할 때 element를 제대로 선언하지 않은 것을 발견했고 이를 수정해서 성공적인 결과 페이지를 얻을 수 있었다.