4주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20211522 이름: 김정환

**1.**

.................

<LinkedList.h>

#ifndef \_\_LINKEDLIST\_\_

#define \_\_LINKEDLIST\_\_

#include <iostream>

using namespace std;

//LinkedList Node

template <typename T>

class Node{

    public:

        //데이터를 저장할 변수

        T data;

        //노드구조체 이용; 다음노드의 주소를 저장할 포인터

        Node<T> \*link;

        Node(T element){

          data = element;

          link = 0;

        }

    };

//LinkedList Class

template <typename T>

class LinkedList{

    protected:

        //첫번째 노드의 주소를 저장할 포인터

        Node<T> \*first;

        int current\_size;

    public:

        //생성자, 초기화

        LinkedList(){

            first = 0;

            current\_size = 0;

        };

        //노드 개수를 리턴

        int GetSize(){

            return current\_size;

        };

        //맨 앞에 원소를 삽입, LinkedList와 Stack 둘다 같다

        void Insert(T element);

        //맨 뒤의 원소를 삭제, 제일 나중에 들어온 원소 삭제  - LinkedList

        virtual bool Delete(T &element);

        //리스트 출력

        void Print();

};

//새 노드를 맨 앞에 붙이고 값을 넣음

template <typename T>

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

    //노드 생성

    Node<T> \*newnode = new Node<T>(element);

    //새 노드가 첫번째 노드를 가리킴

    //newnode는 포인터이기 때문에 -> 를 사용해 함수, 변수를 불러옴 newnode.link와 뜻은 같다

    newnode -> link = first;

    first = newnode;

    current\_size++;

}

//마지막 노드의 값을 리턴하면서 메모리에서 할당 해제

template <typename T>

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

    if (first == 0)

        return false;

    Node<T> \*current = first;

    Node<T> \*previous = 0;

    //마지막 노드까지 찾아가는 반복문

    while(1){

        if (current->link == 0){  //마지막 노드를 찾는것

            if (previous)

                previous -> link = current -> link;

            else

                first = first -> link;

            break;

        }

        previous = current;

        current = current -> link;

    }

    element = current -> data;

    delete current;

    current\_size--;

    return true;

}

//리스트를 출력하는 Print 함수

template <typename T>

void LinkedList<T>::Print(){

    Node<T> \*i;

    int index = 1;

    if (current\_size != 0){

        for( i = first; i != NULL; i=i->link){

            if (index == current\_size){

                cout << "[" << index << "|";

                cout << i -> data <<"]->";

            }

            else {

                cout << "[" << index << "|";

                cout << i -> data << "]->";

                index++;

            }

        }

        cout << endl;

    }

}

#endif

<Stack.h>

#include "LinkedList.h"

//1. 템플릿 클래스로 확장해야함

//2. Stack형식으로 Delete 함수 재정의해야함

//주의: first, current\_size는 class의 멤버 변수이기 때문에 this 포인터를 사용하여 가져와야함

//LinkedList class를 상속받음

template <typename 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

            Node<T>\* cur = this -> first; // 첫번째 노드 변수에 저장

            this -> first = this -> first -> link; // 기존 스택의 첫 번째 노드는 기존의 두 번째 노드로 변경

            element = cur -> data; // 멤버 함수의 인자에 삭제할 노드의 data 값 저장

            delete cur; // 노드의 메모리 해제

            this -> current\_size--; // 스택의 총 개수에서 1 빼주기

            return true; // 삭제 과정 진행이 끝나면 true 반환

            }

};

<main.cpp>

#include <stdio.h>

#include "Stack.h"

void prnMenu(){

    cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

    cout<<"\* 1. 삽입    2. 삭제    3. 출력   4. 종료 \*"<<endl;

    cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

    cout<<endl;

    cout<<"원하시는 메뉴를 골라주세요: ";

}

int main(){

        int mode, selectNumber, tmpItem;

        LinkedList<int> \*p;

        bool flag = false;

        cout<<"자료구조 선택(1: Stack, Other: Linked List): ";

        cin>>mode;

        if(mode == 1)

            p = new Stack<int>();    // 정수를 저장하는 스택

        else

            p = new LinkedList<int>();

        do{

            prnMenu();

            cin>>selectNumber;

            switch(selectNumber){

                case 1:

                    cout<<"원하시는 값을 입력해주세요: ";

                    cin>>tmpItem;    p->Insert(tmpItem);

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

                    break;

                case 2:

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

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

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

                    break;

                case 3:

                    cout<<"크기: "<<p->GetSize()<<endl;

                    p->Print();

                    break;

                case 4:

                    flag = true;     break;

                default:

                    cout<<"잘못 입력하셨습니다."<<endl;

                    break;

            }

            if(flag) break;

        } while(1);

        return 0;

}

<알고리즘과 자료구조 관련 설명>

LinkedList는 class로 Node와 LinkedList를 선언하여 구현하였다. Node 클래스에는 데이터를 저장할 변수 data를 template 사용하여 선언하고, 다음 노드의 주소를 저장할 Node 클래스의 객체의 포인터 변수 link를 선언하고, 생성자를 템플릿을 사용한 타입을 가진 element 변수를 인자로 받아 data에 element 값을 저장하고 link의 주소값을 0으로 초기화하도록 했다. LinkedList 클래스에는 protected 접근지정자를 가진 첫 노드의 주소를 저장할 first, 연결리스트의 노드 총 개수를 저장할 int 타입의 current\_size 변수를 선언했다. 생성자에서는 firtst와 current\_size를 모두 0으로 초기화하였다. 멤버 함수 int GetSize()는 current\_size를 return함으로써 외부에서도 값을 획득할 수 있게 하였다. 다음으로 연결리스트에서 값을 추가하는 void Insert(T element) 함수이다. 실습에서는 연결리스트에서 무조건 제일 앞에만 노드가 추가되도록 구현하는 것이 목표이므로 해당 함수는 제일 앞에 노드를 추가하는 기능을 가진다. 새로운 노드는 함수의 인자로 받은 element 값을 인자로 전달해주며 newnode 변수에 동적할당해준다. 그 후 기존의 LinkedList 클래스의 객체의 first 변수에 저장되어 있던 노드가 newnode 다음에 오도록 newnode->link에 first를 대입한다. 그 후 first에 newnode를 대입하여 첫 노드가 바뀌도록 하고 current\_size를 1 증가시켜 총 개수가 늘어나도록 한다. bool Delete(T &element) 함수는 마지막 노드의 값을 인자로 받은 주소에 전달하며 노드를 삭제하는 함수이다. first가 0인 경우에는 리스트에 값이 하나도 없는 경우이므로 false 값을 리턴하고 종료한다. 그리고 마지막 노드를 가리키게 할 current와 그 이전 노드를 가리키게 할 previous 변수를 선언한다. 그 후 current->link가 0이 될 때까지 previous에 current의 노드를 옮기고, current에는 current->link가 가도록 한다. current->link가 0이 되면 previous가 0이 아닐 때는 previous->link에 current->link가 가도록 하고, 빈 값인 경우에는 리스트에 값이 하나만 있는 것이므로 first는 first->link로 옮긴다. 그 후 함수의 인자로 받은 element에 current->data의 값을 전달하고, delete를 이용해 current의 메모리를 해제한다. 또한 current\_size를 1 줄여 총 개수도 하나 줄어들도록 한 후 true값을 리턴한다. void Print() 함수는 current\_size가 0이 아닌 경우만 리스트의 끝까지 탐색하며 주어진 형식과 맞게 출력하도록 하였다.

Stack class는 마찬가지로 템플릿을 사용하여 저장하는 데이터의 타입을 전달하였고, LinkedList class를 상속받았다. Stack class에는 bool Delete(T &element) 함수만 재정의하였다. Stack은 LinkedList와 다르게 LIFO(Last In First Out) 방식이므로 first가 가리키는 노드가 삭제되어야 한다. 또한 상속받은 클래스의 멤버 변수에 접근할 때 필요한 this를 사용하였고, first가 0일 때는 아무 데이터도 없으므로 false를 리턴하며 함수가 종료된다. 데이터가 있을 때는 우선 first가 가리키는 노드를 cur 변수에 저장해주고, first는 기존의 first->link가 가리키는 노드로 대입하였다. 인자로 받은 element에 기존 첫 번째 노드의 data를 넣어주어 값을 반환해주고, cur의 메모리를 delete를 사용하여 해제하였다. 또한 current\_size를 1 줄여 개수도 하나 줄도록 하였다. 과정이 끝나면 true를 리턴해주며 함수가 종료된다.

<실습 결과화면>

텍스트, 스크린샷, 흑백, 메뉴이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 흑백, 블랙이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 블랙, 회로이(가) 표시된 사진

자동 생성된 설명

................