1. 실험 시간에 작성한 프로그램의 알고리즘과 자료구조

(1) LinkedList.h

LinkedList.h에서는 Node 클래스와 LinkedList 클래스를 구현했다. 링크드 리스트에 사용될 개별 노드는 별도의 Node 클래스를 사용한다. Node 클래스는 여러 가지 자료형에 대해 사용 가능하도록 템플릿 클래스로 작성했고, public으로 T형 변수 data, 다음 노드를 포인트하는 포인터 변수 link가 선언되어 있다. LinkedList 클래스에서는 노드와 링크드 리스트의 크기를 저장하는 정수형 변수를 protected로 가진다. 멤버 함수로는 GetSize(), Insert(), Delete(), Print() 함수를 가진다. GetSize()에서는 protected로 보호하고 있는 링크드 리스트의 크기 값을 반환하고, Insert()에서는 링크드 리스트의 맨 앞에 노드를 삽입하며, Delete()에서는 링크드 리스트 맨 뒤에 있는 노드(가장 마지막에 들어온 노드)를 삭제한다. 이 때 Stack 클래스에서 LinkedList 클래스를 상속받으며 Delete() 함수를 overriding할 수 있도록 Delete() 함수는 virtual로 선언되었다. Print()에서는 링크드 리스트의 내용을 출력한다. 함수 내부의 설명은 주석으로 작성하였다.

```
#ifndef __LINKEDLIST__
#define __LINKEDLIST__

#include <iostream>
using namespace std;

//LinkedList Node
//다양한 자료형에 대해 사용 가능하도록 템플릿 클래스로 작성
template <typename T>
class Node{
    public:
        T data; //데이터를 저장할 변수
        Node *link; //노드 구조체 이용; 다음 노드의 주소를 저장할 포인터

        //생성자, 초기화
        Node(T element){
          data = element;
          link = 0;
        }
    };

//LinkedList Class
//다양한 자료형에 대해 사용 가능하도록 템플릿 클래스로 작성
template <typename T>
class LinkedList{
    protected:
        Node<T> *first; //첫번째 노드의 주소를 저장할 포인터
        int current_size; //링크드리스트의 크기를 저장할 변수
    public:
        //생성자, 초기화
        LinkedList(){
```

```
            first = 0;
            current_size = 0;
        };

        //노드 개수를 리턴
        int GetSize(){
            return current_size;
        };

        //맨 앞에 원소를 삽입, LinkedList와 Stack 둘 다 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){
    Node<T> *newnode = new Node<T>(element); //노드 생성

    //newnode는 포인터이기 때문에 -> 를 사용해 함수, 변수를 불러옴
    //newnode.link와 뜻은 같음
    newnode -> link = first; //새 노드가 첫번째 노드를 가리킴
    first = newnode; //새 노드를 첫번째 노드로 만듦(맨 앞에 추가함)
    current_size++; //링크드리스트 크기를 1 증가
}

//마지막 노드의 값을 리턴하면서 메모리에서 할당 해제
template <typename T>
bool LinkedList<T>::Delete(T &element){
    //링크드리스트가 비어있다면(first가 0이라면) false를 반환
    if (first == 0)
        return false;

    Node<T> *current = first; //current를 첫번째 요소로 설정
    Node<T> *previous = 0; //previous는 0으로 초기화

    //마지막 노드까지 찾아가는 반복문
    while(1){
        //current가 마지막 노드라면, previous의 존재 유무에 따라 다른 실행을 하고
루프 종료
        if (current->link == 0){
            //previous가 있다면, previous가 마지막 요소가 될 것이므로 previous-
>link를 current->link로 설정
            if (previous)
```

```
                previous -> link = current -> link;
            //previous가 없다면, current가 유일한 요소이므로 first를 first-
>link로 리셋
            else
                first = first -> link;
            break;
        }
        //다음 노드로 넘어감
        previous = current;
        current = current -> link;
    }
    element = current -> data; //전달받은 인자에 삭제 대상 노드의 데이터를 저장
    delete current; //노드 메모리 할당 해제
    current_size--; //링크드리스트 크기 1 감소

    return true;
}

//리스트를 출력하는 Print 함수
template <typename T>
void LinkedList<T>::Print(){
    Node<T> *i;
    int index = 1;

    //링크드리스트에 있는 노드가 1개 이상일 때만 출력
    if (current_size != 0){
        for( i = first; i != NULL; i=i->link){
            //마지막 노드는 "]"까지 출력
            if (index == current_size){
                cout << "[" << index << "|";
                cout << i -> data <<"]";
            }
            //마지막 노드가 아닌 노드들은 "]->"까지 출력
            else {
                cout << "[" << index << "|";
                cout << i -> data << "]->";
                index++;
            }
        }
        cout << endl;
    }

}


#endif
```

(2) Stack.h

Stack.h에서는 LinkedList 클래스를 상속받는 Stack 클래스를 구현했으며, Stack 클래스는 LinkedList 클래스와 마찬가지로 템플릿 기반의 클래스로 구현했다. LinkedList의 멤버 함수 중 Delete() 함수만 overriding(재정의)해서 맨 뒤의 노드가 삭제되는 링크드 리스트와 달리 스택에서는 맨 앞의 노드가 삭제되도록 한다. 함수 내부의 설명은 주석으로 작성했다.

```
#include "LinkedList.h"

//LinkedList class를 상속받음
//다양한 자료형에 대해 사용 가능하도록 템플릿 클래스로 작성
template <typename T>
class Stack : public LinkedList<T>{
    public:
        //Stack은 LinkedList와 달리 가장 첫 요소를 삭제하므로 delete() 함수만
overriding
        bool Delete (T &element){
            //스택이 비어있다면(first가 0이면) false반환
            if(this->first == 0) {
                return false;
            }

            Node<T> *current = this->first; //current를 스택의 첫 요소로 설정

            this->first = current->link; //스택의 첫 요소를 첫 요소의 link로
설정


            element = current->data; //전달받은 인자에 삭제 대상 요소의 데이터를
저장
            delete current; //메모리 할당 해제
            this->current_size--; //스택 크기 1 감소

            return true;
            }
};
```

(3)-1 템플릿 클래스로 변경하여 출력 결과를 확인하는 main.cpp

double 자료형을 저장하는 링크드 리스트, string 자료형을 저장하는 링크드 리스트를 만들고 각각에 대해 삽입, 삭제, 출력 기능을 확인했다.

```
#include <stdio.h>
#include <string>
#include "Stack.h"

int main() {
    double dVal;
    string strVal;

    LinkedList<double> dList;
    LinkedList<string> strList;

    /* double 자료형 링크드리스트*/
    //삽입
    dList.Insert(3.14);
    dList.Insert(123456);
    dList.Insert(-0.987654);
    //출력
    dList.Print();
    //삭제 및 확인
    dList.Delete(dVal);
    cout<<"삭제된 마지막 원소: "<<dVal<<endl;
    //출력
    dList.Print();
    //삽입
    dList.Insert(777.777);
    //출력
    dList.Print();
    //삭제 및 확인
    dList.Delete(dVal);
    cout<<"삭제된 마지막 원소: "<<dVal<<endl;
    //삭제 및 확인
    dList.Delete(dVal);
    cout<<"삭제된 마지막 원소: "<<dVal<<endl;
    dList.Print();
    //삭제 및 확인
    dList.Delete(dVal);
    cout<<"삭제된 마지막 원소: "<<dVal<<endl;
    dList.Print();

    /* string 자료형 링크드리스트*/
    //삽입
    strList.Insert("This");
    strList.Insert("is a");
    strList.Insert("Template");
    strList.Insert("Example");
```

```
    //출력
    strList.Print();
    //삭제 및 확인
    strList.Delete(strVal);
    cout<<"삭제된 마지막 원소: "<<strVal<<endl;
    //삽입
    strList.Insert("Class");
    //출력
    strList.Print();

    return 0;
}
```

(3)-2 최종 테스트 코드 main.cpp

스택을 사용할 것인지, 링크드 리스트를 사용할 것인지 입력받은 뒤, 입력받은 값에 따라 Stack이나 LinkedList 객체를 생성했다. 그리고 삽입-삭제-출력-종료 메뉴를 출력하고 입력받은 값에 따라 삽입-삭제-출력-종료를 실행했다. 종료를 입력받은 것이 아닌 동안 메뉴를 출력하고 입력받는 과정을 반복했다. 삽입 시에는 Insert() 함수를, 삭제 시에는 Delete() 함수를, 출력 시에는 GetSize() 함수를 사용했고, 삽입 및 삭제 시에는 어떤 값이 삽입 및 삭제되었는지 출력해주었다.

```
#include <stdio.h>
#include <string>
#include "Stack.h"

//메뉴를 출력하는 함수
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; //종료 신호를 저장하는 변수

        //사용할 자료구조를 입력받음
        cout<<"자료구조 선택(1: Stack, Other: Linked List): ";
        cin>>mode;

        //Stack
        if(mode == 1)
            p = new Stack<int>(); // 정수를 저장하는 스택
```

```
        //Linked List
        else
            p = new LinkedList<int>(); //정수를 저장하는 링크드 리스트


        do{
            //삽입-삭제-출력-종료 메뉴 출력
            prnMenu();
            cin>>selectNumber;

            switch(selectNumber){
                case 1:
                    //삽입
                    cout<<"원하시는 값을 입력해주세요: ";
                    cin>>tmpItem;
                    p->Insert(tmpItem);
                    cout<<tmpItem<<"가 삽입되었습니다."<<endl;
                    break;

                case 2:
                    //삭제
                    if(p->Delete(tmpItem)==true)
                        cout<<tmpItem<<"가 삭제되었습니다."<<endl;

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

                    break;

                case 3:
                    //출력
                    cout<<"크기: "<<p->GetSize()<<endl;
                    p->Print();
                    break;

                case 4:
                    //종료 (flag을 true로 변경)
                    flag = true;
                    break;

                default:
                    //예외 처리
                    cout<<"잘못 입력하셨습니다."<<endl;
                    break;

            }

            if(flag) break;

        } while(1);

        return 0;
}
```