컴퓨터공학실험1 4주차 결과보고서

전공: 컴퓨터공학과 학년: 2 학번: 20191559 이름: 강상원

문제 1. 실습 코드를 word 파일로 복사해서 주석을 첨부하여 설명할 것.

<LinkedList.h>

```
#ifndef __LINKEDLIST__
#define __LINKEDLIST__

#include <iostream>
using namespace std;
////
//template <typename T> // 템플릿 설정
////

//LinkedList Node
template <class T> class Node{
	public:
		//데이터를 저장할 변수
		T data; // 노드의 값 저장
		//노드구조체 이용; 다음노드의 주소를 저장할 포인터
		Node *link;

		Node(T element){
		   data = element;
		   link = 0;
		}
	};

//LinkedList Class
template <class T> class LinkedList{
	protected:
		//첫번째 노드의 주소를 저장할 포인터
		Node<T> *first;
		int current_size; // 링크드 리스트의 크기를 저장
	public:
		//생성자, 초기화
		LinkedList(){
			first = 0;
			current_size = 0;
		};

		//노드 개수를 리턴 (=current size)
		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); // 마지막 원소 제거, 처음부터 끝까지 탐색->시간복잡도 O(n).

		//리스트 출력
		void Print(); //링크드 리스트의 모든 원소 차례로 출력->시간복잡도 O(n).
};


//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) //링크드 리스트 : 0개라면 FALSE.
		return false;

	Node<T> *current = first;
	Node<T> *previous = 0;

	//마지막 노드까지 찾아가는 반복문
	while(1){ // 선형 탐색으로 링크드 리스트의 마지막 원소를 찾는다.
		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--; // 원소의 개수 1 감소.

	return true; // 성공적으로 종료.
}

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

	if (current_size != 0){ // 원소의 개수가 0이 아닐 때만 출력.
		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" // 만든 링크드 리스트 헤더 파일 포함시키기.

//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> // 템플릿 클래스
class Stack:public LinkedList<T> {
public:
	bool Delete(T &element); // 맨 뒤의 원소를 삭제
};

template <typename T>
bool Stack<T>::Delete(T &element) { // Delete 함수 재정의
	if(this->first == NULL) return false; // this 포인터 이용, 자가 참조.
	element = this->first->data;
	Node<T> *t = this->first;
	this->first = this->first->link;
	delete t;
	this->current_size--;
	return true;
}
```

<main.cpp>

```
// 만든 헤더파일들 포함시키기.
#include <stdio.h>
#include "LinkedList.h"
#include "Stack.h"

// 옵션 텍스트 출력.
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;
			// 선택: 1-Stack, 2-Linked List.
			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>();

			// do-while문으로 일단 한번 실행하고 탈출 조건 나올 때까지 반복
			do{
				prnMenu();
				cin>>selectNumber;

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

					case 2: // 삭제 (Delete 함수 이용)->비어 있을 때는 삭제 실패.
						if(p->Delete(tmpItem)==true)
							cout<<tmpItem<<"가 삭제되었습니다."<<endl;

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

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

					case 4: // 종료 조건->while 문 탈출.
						flag = true;	break;

					default: // 잘못 입력했을 때
						cout<<"잘못 입력하셨습니다."<<endl;
						break;

				}

				if(flag) break;

			} while(1);

			return 0;
}
```

문제 2. 다형성, 캡슐화, 재정의를 명확하게 설명할 것

- 다형성

  변수, 상수, 함수, 오브젝트 등의 요소가 자료형에 관계없이 작동되는 성질을 의미한다. 다형성은 크게 서브타입 다형성(subtype polymorphism), 변수 다형성(parametric polymorphism), 임시 다형성(ad hoc polymorphism)이 있다.

- 캡슐화

  객체의 일부를 private 처리하여 구현 내용 일부를 외부에서 접속할 수 없게 만드는 것이다. 데이터와 그 함수를 연계하여 묶는다. 캡슐화의 이점은 다른 코드에서 필요한 부분만 빼서 편이하게 쓸 수 있고, 객체 내부 데이터가 임의로 수정되는 것을 방지할 수 있다는 것이다.

- 재정의

  상위 클래스에서 상속받은 함수 등 메서드를 하위 클래스에서 재정의하는 것을 의미한다.

문제 3. 이번 4주차 실습에서 진행한 코드에서 서브타입 다형성이 적용되는 부분을 명시하고 그 이유를 기술할 것 (4주차 강의 PPT 참고)

```
void prnMenu()
{
        cout<<"*********************************************"<<endl;
        cout<<"* 1. 삽입     2. 삭제     3. 출력     4. 종료 *"<<endl;
        cout<<"*********************************************"<<endl;
        cout<<endl;
        cout<<"원하시는 메뉴를 골라주세요: ";
}

int main()
{
        // 스택 및 연결 리스트 테스트용 코드
        int mode, selectNumber, tmpItem;
        LinkedList<int> *p;
        bool flag = false;

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

        // 기반 클래스의 포인터를 사용하여 기반 클래스 뿐만 아니라
        // 파생 클래스의 인스턴스 또한 접근할 수 있다.
        if(mode == 1)
                p = new Stack<int>();          // 정수를 저장하는 스택
        else
                p = new LinkedList<int>();     // 정수를 저장하는 연결 리스트
```

mode가 1일 경우 서브타입 다형성이 구현된다. LinkedList 클래스를 Stack 클래스가 상속받기 위해서 기반 클래스의 포인터에 파생 클래스 인스턴스의 주소를 저장할 수 있게 한다.

문제 4. 실습 문제 및 과제에 대한 해결방법으로 자료구조 및 알고리즘을 기술 할 것 (프로그램 구조도 반드시 첨부!)

- **실습 문제**

  클래스 사용해 Node, LinkedList 정의했다. 이 상위 클래스를 상속받아 Stack 클래스를 정의한다. -> 문제 의도대로 삽입, 제거, 출력을 하는 프로그램 제작

  template <class T> class Node

  ➢ public

  - T data : 노드의 값 저장
  - Node *link : 다음 노드의 주소를 저장할 포인터

  template <class T> class LinkedList

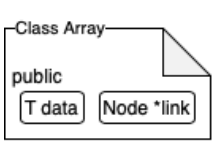
  ➢ protected

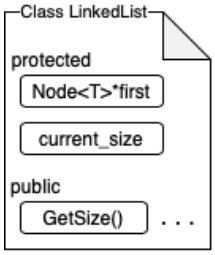
  - Node<T> *first : 링크드 리스트의 첫 노드 참조 저장
  - int current_size : 링크드 리스트의 크기(길이)

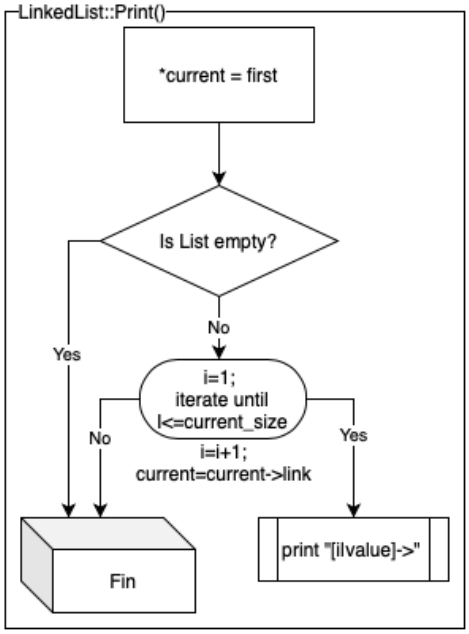
  ➢ public

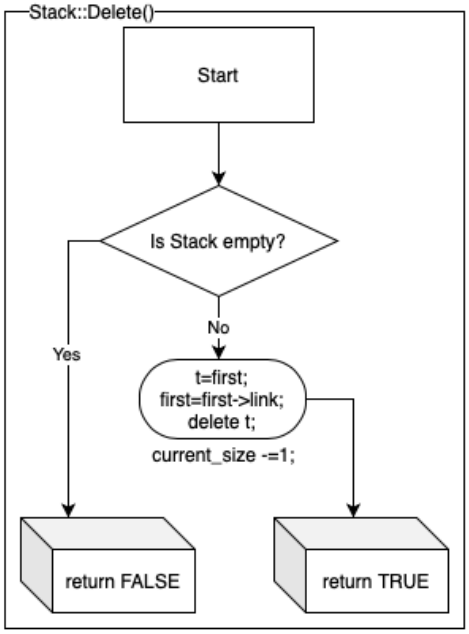
  - int GetSize() : 링크드 리스트의 크기 반환 (protected인 current_size 반환)
  - void Insert(T element) : 링크드 리스트 맨 앞에 원소 저장. 새 노드 생성하여 첫 번째 노드로 설정.
  - virtual bool Delete(T &element) : 링크드 리스트의 맨 끝 (마지막) 원소 제거. 선형 탐색으로 끝까지 탐색하여 마지막 노드를 찾아낸 후, 제거-> current_size도 1 줄인다.
  - void Print() : 선형 탐색으로 링크드 리스트의 모든 원소 순차적으로 출력.

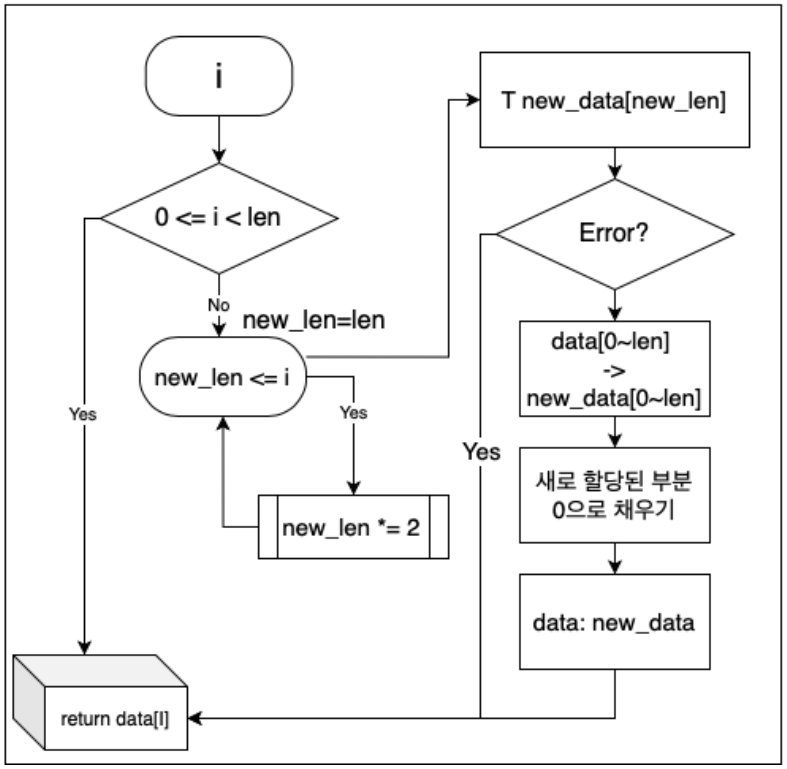
구조도 다음 쪽에 첨부 ->











과제 자료구조, 알고리즘 다음 쪽에 ->

- **과제**

  RangeArray를 만들 때 사용했던 클래스를 상속받아 사용한다. 기존 RangeArray 클래스와 상속받은 Growable Array 클래스의 차이는 operator이다. 배열을 사용하여 입력하는 데이터를 체계적으로 정리하고 사용하게끔 한다. 임의의 자료형을 저장할 수 있게 하기 위해 Array 클래스를 템플릿을 사용하여 확장한다.

  template <class T> class Array

  ➢ protected

  - T *data : 배열 값 저장
  - int len : 배열 길이

  ➢ public

  - Array(){} : 생성자 (배열)
  - Array(int size) : 길이가 size인 배열 생성
  - ~Array() : 소멸자 (배열)
  - int length() const : 배열 길이(len) 반환
  - virtual T &operator[](int i) : 배열의 i번째 원소의 포인터 반환-> 범위 벗어날 시, 오류
  - virtual T operator[](int i) const : 배열의 i번째 원소의 값 반환-> 범위 벗어날 시, 오류
  - void print() : 배열 내의 원소 순차적으로 모두 출력

  template <class T> class GrowableArray : public Array<T>

  ➢ public

  - GrowableArray(int size) : 길이 size인 배열 생성하는 생성자
  - virtual T &operator[](int i) : 배열의 i번째 원소의 포인터 값 반환