4주차 결과보고서

전공: 수학과,컴퓨터공학과 학년: 3학년 학번: 20181256 이름: 김도현

1.1

이번 주차에서는 c++이 가지는 여러 특성들을 알아보기 위해 LinkedList(queue) 형태로 작동하는 기반클래스를 만들고 이를 상속하는 Stack 파생 클래스를 작성해 보았다. 이때 기본 파일이 int 데이터 형을 받아오기 때문에 여러 데이터를 받아올 수 있도록 template 을 사용하였다.

먼저 1번 실험은 LinkedList가 int 데이터 형만 아니라 여러 데이터 형(여기서는 double, string 형)을 받을 수 있도록 수정해보는 것이다. 이때 string은 #include <string>을 헤더 삽입부에 적어 string 데이터 형을 사용할 수 있도록 한다. 즉 template를 사용하여 파라미터적 다형성을 알아보는 실험이다. 1번 실험에 관한 코드 설명이다.

LinkedList.h //LinkedList 를 구현하는 헤더파일이다.

template<class T> class Node

Node 클래스 선언을 할 때 template<class T> 를 적어줘 int형 뿐만 아니라 다른 데이터 유형 역시 받을 수 있도록 템플릿 T(자료형)을 변수들에 사용 할 수 있도록 한다.

여기에는 접근지정자를 public으로 하여 Node의 값을 저장하는 T data, 다음 Node 를 연결할 때 주소를 저장하는 Node \*link, Node(T element) 함수에서는 받아오는 element를 data에 넣고 첫 link는 0으로 한다.

template<class T> class LinkedList

역시 template<class T> 를 클래스 선언 앞에 적어 템플릿자료형을 사용할 수 있도록 한다. 이후함수 코드에서 데이터 형을 T로 바꿔주면 된다. 멤버 변수는 접근지정자를 protected로 하여 후에 사용할 파생클래스에서도 사용 가능하도록 한다.

Node<T> \*first //LinkedList 의 first 노드 주소를 저장하는 변수다.

int current\_size // 현재 LinkedList 의 크기를 int 형으로 저장한 변수다.

이후 멤버 함수는 public으로 접근지정자를 둔다.

LinkedList()

생성자로 Node<T> \*first와 int current\_size를 초기화 한다.

int GetSize()

current\_size 즉 현재 노드의 개수를 return 한다.

void Insert(T element)

새로운 노드를 생성하여 LinkedList의 가장 앞에 넣는 함수로 new를 이용하여 받아온 element가data인 newnode를 생성하고 newnode의 link를 기존의 first로 연결한다. 이후 first를 newnode로 재지정하여 가장 앞 노드가 newnode 가 되도록 한다. 그리고 current\_size도 1 늘린다.

virtual bool Delete(T &element)

2번 실험에서 진행할 Stack 클래스를 생성할 때 Delete만 재정의하면 되기 때문에 같은 의미에서도 객체 스스로 다른 결과를 나오게 하는 서브타입 다형성 개념을 쓰기 위해 동적 타입을 의미하는 함수인 virtual(가상 함수)을 적었다.

만약 first가 0 이면 아무것도 하지 않고 false를 return 한다.

여기서는 현재노드를 의미하는 current 노드와 이전노드를 의미하는 previous 노드를 만들어 queue형태인(first in first out) 가장 처음 들어온(마지막) 노드까지 접근하여 삭제해야 한다. while 반복문을 통해 current->link 가 0일 때 까지 previous=current, current=current->link 로 계속 마지막 노드를 찾는다. 이후 current->link가 0이고 previous가 존재한다면 (노드가 1개 있을 때는 previous 없음) previous->link=current->link로 하고 previous가 0이면 first를 first->link로 지정한다. 그리고 element 에 current(가장 마지막 노드)의 data를 넣고 current 를 삭제 시킨다. 또한 current\_size도 1개 줄인다. 이때는 true를 return 하여 성공적으로 마지막 노드를 delete 했음을 알린다.

void Print() (time complexity O(n))

여기서는 LinkedList를 알맞는 형식에 노드들의 원소를 추출하여 출력하는 함수인데 Node<T> \*i를 선언하여 current\_size가 0이 될때까지 for 문을 통해 노드의 data를 추출한다. 이때 node는 n 개 라면 time complexity O(n)이다.

main.cpp

실험 1에서 main.cpp는 여러 타입의 데이터 형을 잘 받아들이는 지 확인하는 코드이다. 우선 double 형인 dList를 만들고 double 값들이 잘 입력되고 출력되는지 확인한다. 그리고 string형인 strList를 만들어 여기에는 string 값들이 잘 입력되고 출력되는지 확인한다. Print를 통해 입력 값들이 잘 들어왔고 delete를 통해 삭제 역시 잘 되는 것을 알 수 있었다.

이때 string은 #include <string>을 헤더 삽입부에 적어 string 데이터 형을 사용할 수 있도록 한다.

1.2

2번 실험에서는 LinkedList를 상속하는 Stack 클래스를 구현하는 실험이다. 모든 기능을 그대로 가져다 쓰되 위에 Delete 함수에서 언급했다시피 Stack은 last in first out 형태로 마지막에 들어온 노드가 처음으로 삭제되어야 한다. 즉 Delete 함수를 다시 정의(Overriding)하면 된다. 이미 LinkedList의 Delete는 virtual 함수이기 때문에 서브타입 다형성을 이용할 수 있다. 그리고 모호성을 없애기 위해 this 포인터를 이용해서 first나 current\_size 같은 class의 멤버 변수를 이용한다.

template<Class T> class Stack : public LinkedList<T>

템플릿 클래스로 확장하고 LinkedList를 상속한다.

여기서 this->first가 0이면 노드가 없기 때문에 false를 return 한다. 그리고 current노드를 만들어this->first로 두고 this->first=current->link 즉 마지막에서 2번째로 들어온 노드로 first를 재지정한다. 또한 element를 current->data로 한다.(첫번째 노드) 이후 current 를 삭제하고 this->current\_size 역시 1 줄인다. 그리고 성공적으로 삭제했기 때문에 true를 return 한다.