6주차 결과보고서

전공 : 컴퓨터공학 학년 : 2학년 학번 : 20151616 이름 : 최승환

아래에 보인 사항을 작성하여 다음 실험 시간에 제출하시오.

1. 실험 시간에 작성한 프로그램의 알고리즘과 자료구조를 요약하여 기술하시오.

이번 실험은 기존에 주어진 LinkedList class와 Stack class를 템플릿을 이용해 여러가지 타입을 받을 수 있게 만드는 것이다. 이 경우 자료구조는 그대로 유지하지만 템플릿을 적용해 주기 위해 코드 일부를 추가하고 고쳐야 한다. LinkedList 의 특성상 링크를 타고 하나씩 출력하는 방식을 사용하기 때문에 Print함수에서 이 점을 특히 유의해야 한다. c에서 사용하던 printf 함수를 사용하게 되면 %d나 %c와 같은 방식으로 출력을 하게 되는데 템플릿을 사용한 경우 이 방식을 사용하는데에 곤란을 겪을 수 있다. 타입이 다를 때마다 다른 출력 방식을 적용할 수는 없기 때문이다. 이 때문에 iostream 라이브러리에 있는 cout 함수를 사용했다. cout을 사용할 경우 출력 방식을 바꿀 필요 없이 여러 타입을 출력할 수 있다. 따라서 cout을 이용해 Print함수를 사용한다. 이 외에 Insert와 Delete 함수 등에서는 template<typename T>를 이용해 템플릿을 적용해준다. 단, 이를 적용할 시에 current\_size까지 템플릿으로 바꾸는 경우가 있는데 이는 링크드리스트의 노드 개수이므로 정수임이 맞다. 이를 수정해서는 안된다.

알고리즘

LinkedList class

생성자

우선 생성자는 초기화를 해야하므로 데이터를 모두 0(NULL)으로 해주면 된다.

void Insert(T element)

LinkedList에 값을 넣는 함수이다. Node<T>의 포인터를 생성하고 공간 할당을 하고 새롭게 생성한 포인터가 first를 가리키게 하고 first를 새롭게 생선한 포인터를 가리키게 하면 된다. 또한 값을 넣어서 current\_size를 하나 증가시켜주어야 한다.

virtual bool Delete(T &element)

LinkedList 클래스에서는 QUEUE 방법을 따르기 때문에 선입선출 즉 먼저 들어온 것이 먼저 나가는 방법으로 짜야한다. 우선 first가 0이면 false를 리턴하고, 0이 아니면 현재의 위치를 가리키는 포인터 current와 그 전을 가리키는 포인터 previous를 생성한다. 그리고 while문을 사용해서 맨 먼저 들어온 값을 찾는다. (current->link가 0이면 맨 처음 들어온 값이다) 찾은 후에는 previous->link를 0을 가리키게 만들고 (하나의 값만 있는 경우에는 first를 0을 가리키게 만든다) 값을 element에 저장하고 delete하면 된다. 또한 값을 삭제했기 때문에 current\_size를 하나 감소시켜주어야 한다. 그리고 main 함수에서 LinkedList 클래스 포인터로 Stack 객체를 가리키게 하는데 이 때 LinkedList 클래스와 Stack 클래스의 삭제하는 방법이 다르다. 하지만 문제에서 의도하는 대로 포인터가 가리키는 객체의 자료형의 함수를 실행해야하기 때문에 virtual을 사용해서 Stack 클래스에서 오버라이딩이 된 함수를 사용할 수 있게 만들었다.

ⅰ Stack class

Insert 함수

template <typename T>

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

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

newnode->link = first;

first = newnode;

current\_size++;

}

Delete 함수

template <typename T>

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

if(first == 0) return false;

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

while(1){

if(current->link == 0) // find end node

{

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;

}

Print 함수

template<typename T>

void LinkedList<T>::Print()

{

int i;

Node<T>\* stt=first;

for(i=1;i<=current\_size;++i)

{

if(i != 1)

cout << "->";

cout << "[" << i << "|" << stt -> data << "]";

stt = stt -> link;

}

cout << endl;

}

LinkedList class의 경우와 달리 Stack은 마지막으로 들어온 데이터가 먼저 삭제된다. 하지만 LinkedList class를 상속하기 때문에 Delete 함수만 오버라이딩을 이용해 바꾸어 주면 된다.

Stack class의 Delete 함수

template <typename T>

bool Stack<T>::Delete(T &element)

{

Node<T> \*prev=first;

if (first == 0)

{

return false;

}

if(first -> link == 0)

{

element = first -> data;

first = 0;

delete first;

current\_size--;

return true;

}

element = first -> data;

first = first->link;

delete prev;

current\_size--;

return true;

};

2. 숙제 문제를 해결하기 위한 알고리즘 및 자료구조를 요약하여 기술하시오.

Array 클래스는 여러 타입을 받을 수 있게 템플릿을 적용한다. 그리고 T(템플릿 이름)\*형으로 data를 저장해 T형을 담을 자료구조를 만든다. 이후에 사이즈가 입력될 경우 data = new T[size] 를 이용해 T 배열을 만들어 시작 주소를 data에 저장해준다. 이렇게 하면 어떤 자료형이 들어오던 간에 상관 없이 해당 배열을 만들어줄 수 있다. 또 T& operator[](int i)를 이용하면 T 배열이 시작하는 주소를 넘겨줄 수 있다. 이를 이용하면 array[3]과 같은 문법으로 사용할 수 있다. 그리고 이를 이용할 때는 에러가 발생했을 때 리턴값을 고려해주어야 함을 잊지 말자. 이를 위해 선언하는 변수가 static T temp이다.

GrowableArray class는 Array class를 상속해 대부분의 기능을 그대로 사용하면 된다. GrowableArray 에서는 len과 입력된 size를 비교해 size가 더 큰 경우 data의 크기를 기존의 len의 두 배로 만들어주면 된다. 이를 구현하기 위해 GrowableArray 의 operator[] 함수는 len 보다 size가 더 큰 경우 len의 두 배 크기의 T 배열을 새로 만들어준다. 그리고 이 주소를 반환해준다. 꼭 기존의 배열의 크기를 늘려줄 필요는 없기 때문에 위와 같이 두 배 크기의 새로운 배열을 선언한 후 이를 반환해주는 것도 좋은 방법이라고 할 수 있다. 이 때 기존의 data를 반드시 delete해줘야 한다.

3. 작성한 프로그램을 공지한 제출요령에 의거하여 기한 내에 제출하시오. 요청이 있을 경우 실험 시간에 작성한 프로그램도 아울러 제출한다.