Data structure: Assignment 1

201716472

최유진

1.1 Binary-search

(1)재귀형

코드설계 :

과제의 설명 대로 int BinarySearch(int A[], int start, int end, int key); 함수를 사용하였다.

Int nMid = (start + end) / 2;라는 변수를 그 안에 만들고 그 값과 key의 값을 비교하여 return 값을 BinarySearch 함수로 받음으로써 재귀형으로 만들었다. 이러한 과정으로 key의 숫자가 몇 번째 위치에 있는 지를 나타나게 하였다.

Main() 함수 안에서는 파일 입출력을 위해 fstream을 사용하였다. 그리고 받아낸 txt의 값을 벡터에 저장하고 그 값을 다시 배열에 옮겨 함수에 사용하였다.

결과 :

C:\Users\LG\Pictures\Screenshots\스크린샷(63).png C:\Users\LG\Pictures\Screenshots\스크린샷(66).png

(2)반복형

코드설계 :

재귀형과 비슷한 방식으로 코드를 작성하였다. 하지만 BinarySearch함수 안에 int nEnd = end - 1;을 설정하고 return 값을 start와 nEnd 값으로 받았다. 함수로 반복되어 값이 나오는 재귀형과는 다르게 값이 while문의 조건에 해당하여 while문이 실행되며, key가 몇 번째 위치에 있는 가를 나타나게 하였다.

Main() 함수에서의 파일 입출력과 배열에 숫자를 받는 방식은 재귀형과 동일하다.

결과 :

C:\Users\LG\Pictures\Screenshots\스크린샷(67).png C:\Users\LG\Pictures\Screenshots\스크린샷(68).png

1.2 k-ary search

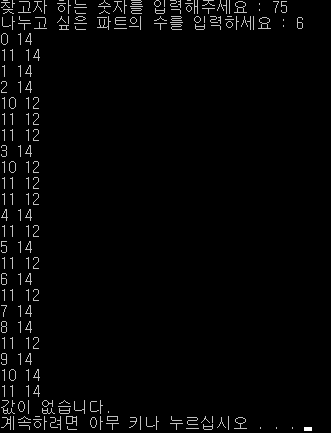
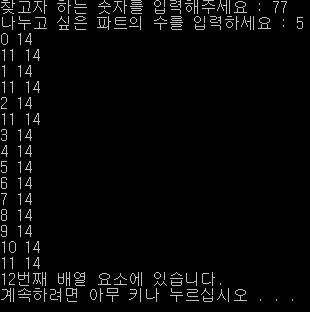
코드설계 :

전체적인 틀은 1.1의 재귀형과 다르지 않으나, int KarySearch(int A[], int k, int start, int end, int key);를 사용하여 몇 파트로 나누는 지를 얻는 int k를 추가하였다.

karySearch 함수에서 for문안에 if(A[start + (end - start) \* i / k] <= key && key <= A[start + ((end - start) \* (i + 1) / k)])라는 조건문을 사용하여 받는 숫자들을 k개의 파트로 나누었고, 그 나눈 파트의 값을 가진 karySearch 함수를 return 값으로 받아 재귀형으로 코드를 만들었다.

Main() 함수에서는 k의 값을 받고, 1.1의 재귀형과 동일하게 파일을 받아 실행하였다.

결과 :



1.3 이중 연결 리스트 구현

1.3.1 이중 연결 리스트 구현 : 정수형 버전

클래스 계층도 :

DoublyListP.cpp

DoublyListP.h

#include “DoublyListP.h”

#include “DoublyListP.h”

DoublyListTest.cpp

코드 설계 :

DoublyListP.h 파일

listClass 안에 결과를 보여주는 void Display(); 함수와 Nptr Tail;을 추가하였다.

DoublyListP.cpp 파일

llistClass() 함수

생성자이다. Head, Tail, Count를 초기화 한다.

listClass(const listClass& L) 함수

Head, Tail, Count를 listClass& L의 Head, Tail, Count로 설정한다.

~listClass() 함수

소멸자. 비어있지 않을 때 Delete를 사용하여 비운다.

Insert(int Position, int item) 함수

집어넣을 숫자(item)과 넣을 위치(Position)를 받아 if문 조건을 부합하지 않으면 else를 실행한다. Nptr p라는 new node를 받아 원하는 위치에 집어넣는다. Count는 1씩 추가된다.

Delete(int Position) 함수

지우고 싶은 위치(Position)를 받아 Insert와 같이 if문 조건이 부합하지 않으면 else를 실행한다. for문을 사용하여 지울 위치의 뒤 값을 지울 위치에 가져온다. 그것을 끝의 값까지 실행한다. Count는 1씩 감소된다.

Retrieve(int Position, int item) 함수

값을 알고싶은 위치(Position)와 알아낼 값을 받을 item을 받는다. Nptr p를 이용하여 Data를 받아 item에 집어넣고 그것을 몇 번째 위치의 무슨 값인지를 출력한다.

Display() 함수

Ex) (10, 20, 30)과 같은 형식으로 출력될 수 있게 하는 함수이다.

isEmpty() 함수

list가 비어있는지를 알려준다.

Length() 함수

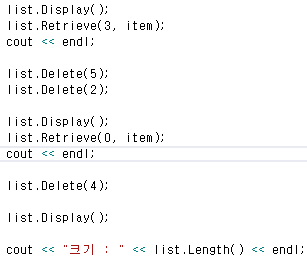
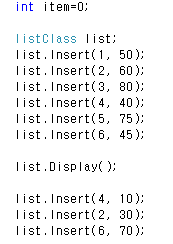
list에 들어있는 숫자의 수를 return 한다..

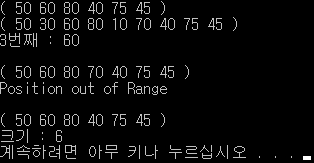
DoublylListTest.cpp 파일

Main 함수안에 Insert, Delete, Retrieve 함수들을 사용하여 list를 채운다.

그리고 Display()를 사용하여 출력하였다.

실행 결과 :

<= 실행될 코드



1.3.2 이중 연결 리스트 구현 : template버전

클래스 계층도

GenericDoublyListP.h

#include “GenericDoublyListP.h”

”

#include “GenericDoublyListP.h”

”

#include “GenericDoubListP.cpp”

”

GenericDoublyListPTest.cpp

GenericDoublyListP.cpp

코드설계 :

GenericDoublyListP.h 파일

template <typename T>를 선언하여 1.3.1의 listClass를 listClass<T>로 바꾸고 Nptr은 nodeRecord<T> \*로 바꾸었다. 1.3.1의 바꿀 수 있는 것들을 모두 T로 바꾸었다.

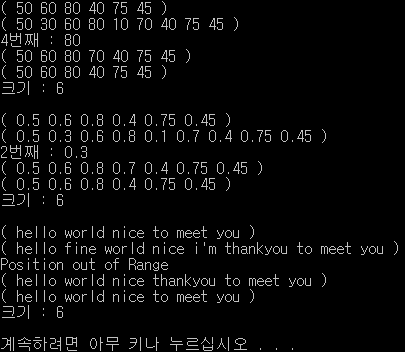
GenericDoublyListP.cpp 파일

GenericDoublyListP.h 파일과 의 설명과 동일하다.

GenericDoublyListPTest.cpp 파일

listClass<int> a1; listClass<double> a2; listClass<string> a3; 을 선언하여 int형 double형 string형의 list를 출력하였다.

실행 결과 :



1.3.3 이중 연결 리스트 구현 – java 버전

코드 설계 :

DoublylinkedList() 함수 :

Head와 count를 초기화 한다.

Class Node :

Object data, Node Prev, Node Next를 선언한다.

Node(Object data) 함수 :

Data를 받아 Object data를 Data로 설정해주고 Prev와 Next를 초기화 한다.

Node getNode(int Position) 함수 :

위치(Position)을 받아 그 위치의 값의 node를 가져온다.

Object get(int Position) 함수 :

getNode에서 받아온 node의 data를 return 한다. getNodeIndex와 함께 Retrieve역할을 한다.

Object getFirst() 함수 :

제일 처음에 위치한 값을 return 한다.

Int getNodeIndex(Object index) 함수 :

Data가 몇 번째에 있는 지를 return 한다.

Void InsertFirst(Object data) 함수 :

Data를 받아 가장 첫 번째 위치에 data를 넣는다.

Void Insert(int Position, Object Item) 함수 :

넣고 싶은 위치(Position)과 넣고 싶은 숫자(Item)을 받는다. Node previous와 Node next를 선언하고, Prev와 Next를 사용하여 원하는 위치에 값을 넣는다. Count는 1씩 증가한다.

Void InsertLast(Object data) 함수 :

Insert에 끝부분 수인 Count와 data를 넣게 하는 함수. 제일 끝부분에 숫자를 Insert되게 한다.

Void Insert(Object data) 함수 :

Insert에 숫자만 넣을 경우 InsertLast 함수를 사용하여 끝부분에 숫자를 넣는 기능을 한다.

Object DeleteFirst() 함수 :

getNode를 사용하여 첫번째 node를 받아 비우고 다음 노드를 firstNode로 받고 그 data를 return 한다. 첫 번째에 위치한 숫자를 지우는 기능을 한다. Count가 1씩 감소한다.

Object Delete(int Position) 함수 :

입력받는 위치(Position)에 있는 숫자를 지우는 기능을 한다. Node deleteNode로 값을 받고 그 값의 다음 번째에 위치한 값을 Node next와 previous를 사용하여 앞으로 가져오는 역할을 한다. deleteNode의 data를 return 한다.

Object DeleteLast() 함수 :

Count-1을 Delete 함수에 넣어 끝부분에 위치한 값을 지운다.

Boolean isEmpty() 함수 :

List가 비어있는지 알려주는 기능을 한다.

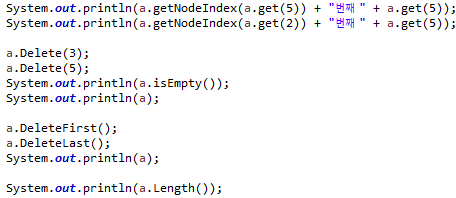
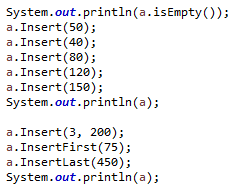
Int Length() 함수 :

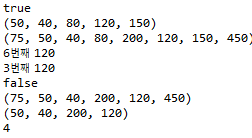
List가 가지고 있는 숫자의 수를 알려준다. Count를 return 한다.

String toString() 함수 :

출력할 숫자들 사이에 “, ”와 “)”, “(“를 출력하게 하는 함수.

실행 결과 :

^ 실행할 test 코드



1.3.4 스택과 큐 구현 : template 이중 연결 리스트 기반

(1) 스택 구현

클래스 계층도 :

#include “genericStackDL.h”

”

stackClassTest.cpp

genericStackDL.cpp

#include “genericStackDL.h”

”

genericStackDL.h

#include “genericStackDL.cpp”

”

코드 설계 :

genericStackDL.h 파일

listClass는 template를 사용하기 위해 1.3.2의 것을 그대로 사용하였다. Template를 사용하며stackClass class를 만들었으며, 사용할 함수들을 선언하였다(선언한 함수들은 genericStackDL.cpp에서 서술함). 사용할 수 있는 모든 곳에 template를 사용하였다. listClass를 사용하기 위해 listClass<T> L;을 선언하였다.

genericStackDL.cpp 파일

listClass의 함수들은 1.3.2와 같다.

stackClass() 함수 :

생성자이다.

stackClass(const stackClass<T>& S) 함수 :

L이 stackClass<T>& S의 L이 되게 한다.

~stackClass() 함수 :

소멸자이다. Pop를 사용하여 stack을 비어있게 한다.

Void Push(T NewItem) 함수 :

listClass의 Insert를 사용하여 받은 Item을 L에 삽입한다.

Void Pop() 함수 :

비어있다면 Stack is empty를 출력하고 그렇지 않다면 Delete를 사용하여 L의 첫 번째에 위치하고 있는 숫자를 지운다.

Void Display() 함수 :

listClass의 Display를 사용한다.

Void GetTop(T& Item) 함수 :

Item을 받아 그 아이템은 listClass의 Retrieve함수에 집어 넣는다. L의 첫 번째에 위치한 숫자가 무엇인지를 알려주는 기능을 한다.

Void Retrieve2(int Position, T& Item) 함수 :

알고 싶은 숫자의 위치(Position)과 Item을 받아 listClass의 Retrieve 함수에 넣어 검색기능을 한다.

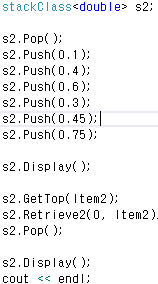
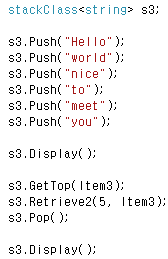
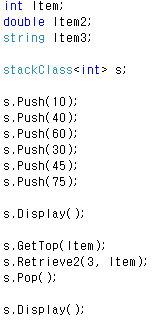
Bool IsEmpty() 함수 :

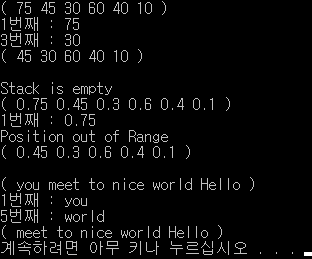
listClass의 IsEmpty()함수를 사용하여 L이 비어있는가를 알려준다.

stackClassTest.cpp 파일

stackClass<int> s; stackClass<double> s2; stackClass<string> s3; 을 선언하고 Push, Pop, GetTop, Retrieve2, Display등의 함수를 이용하여 숫자들을 추가하거나 지우고 찾아내는 기능을 사용하였다.

실행 결과 :

<-실행할 test 코드



(2) 큐 구현

클래스 계층도 :

genericQueueDL.h

#include “genericStackDL.h”

”

queueClassTest.cpp

genericQueueDL.cpp

#include “genericStackDL.h”

”

#include “genericStackDL.cpp”

”

코드 설계 :

genericQueueDL.h 파일

listClass는 stackClass에서 사용한 것과 같다. Template를 사용하며 queueClass를 만들었으며, 사용할 함수들을 선언하였다(선언한 함수들은 genericQueueDL.cpp에서 서술함). 사용할 수 있는 모든 곳에 template를 사용하였다. listClass를 사용하기 위해 listClass<T> Rear;을 선언하였다.

genericQueueDL.cpp 파일

queueClass() 함수 :

생성자이다.

queueClass(const queueClass<T> &Q) 함수 :

Rear이 stackClass<T>& Q의 Rear가 되게 한다.

~queueClass() 함수 :

소멸자이다. Remove를 사용하여 stack을 비어있게 한다.

Void Add(T Item) 함수 :

listClass의 Insert를 사용하여 받은 Item을 Rear에 삽입한다.

Void Remove() 함수 :

비어있다면 Queue is empty를 출력하고 그렇지 않다면 Delete를 사용하여 Rear의 첫 번째에 위치하고 있는 숫자를 지운다.

Void Display() 함수 :

listClass의 Display를 사용한다.

Void GetFront(T& Item) 함수 :

Item을 받아 그 아이템은 listClass의 Retrieve함수에 집어 넣는다. Rear의 첫 번째에 위치한 숫자가 무엇인지를 알려주는 기능을 한다.

Void Retrieve3(int Position, T& Item) 함수 :

알고 싶은 숫자의 위치(Position)과 Item을 받아 listClass의 Retrieve 함수에 넣어 검색기능을 한다.

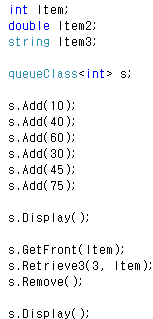
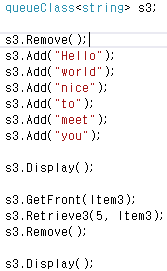
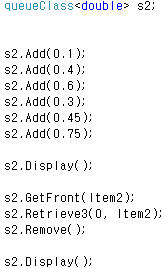
Bool IsEmpty() 함수 :

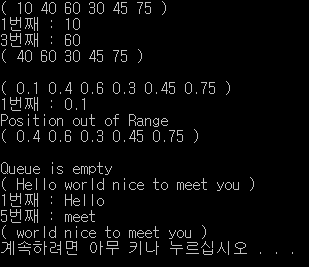
listClass의 IsEmpty()함수를 사용하여 Rear이 비어있는가를 알려준다.

queueClassTest.cpp 파일

queueClass <int> s; queueClass <double> s2; queueClass <string> s3; 을 선언하고 Add, Remove, GetFront, Retrieve3, Display등의 함수를 이용하여 숫자들을 추가하거나 지우고 찾아내는 기능을 사용하였다.

실행 결과 :

 <-실행한 test코드



1.4 산술식 평가 : calculator 구현

코드 설계 :

Int precedence(char op) 함수

계산을 +나 –로 하면 1을 return하고 \*나 /로 하면 2를 return 하는 함수이다.

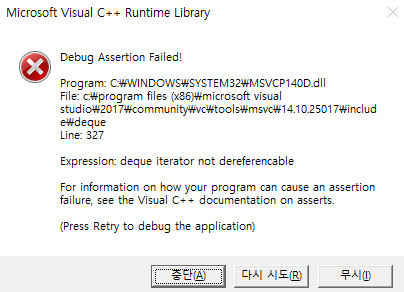
Int calculate(int a, int b, char op) 함수

변수 a, b, op를 받아 op에 따라 a와 b를 계산하여 return 하는 함수이다.

Int calculator (string number)

식을 받아 stack<double> num과 stack<char> ops에 숫자와 기호를 넣어 calculate를 받아 계산하는 함수이다. for문을 사용하여 number의 길이를 받아 “(“와 “ “을 구분하여 number의 배열에 넣었다. 그리고 그것을 stack num과 stack ops에 집어넣었다. 그것을 while을 이용하여 val1과 val2에 각각 넣어 calculate 함수에 숫자 val1와 val2 그리고 기호를 넣어 계산하였다. 그러나 “)”가 잘 구분이 되지 않고 오류가 나 제대로 출력되지 않았다. 그리고 pow나 log같은 기호들을 넣는 것을 실패했다.

실행 결과 :

 C:\Users\LG\Pictures\Screenshots\스크린샷(98).png

1.5 깊이 우선 탐색 : depth-first search 구현

코드 설계 :

int n, m; bool \*\*check, \*visited; stack<int> s; 들을 선언하였다.

Void dfs(int v) 함수

깊이 우선 탐색을 실행시키는 기능을 하는 함수이다. for문을 사용하였다. for문 안에 if(check[v][i] && !visited[i])를 사용하여 깊이 우선 탐색이 실행되도록 만들었으나, 끝부분에서 멈추는 기능을 구현하지 못하였다.

또한 끝부분에 가기까지의 과정 중 끝부분에 관련된 숫자만 출력해야 하는데, 그것을 구현하지 못하여 모든 과정을 출력하였다.

Void input() 함수

dfs에서 사용할 visited와 check 배열들을 true 또는 false로 바꾸는 기능.

Sample\_graph.txt를 1.1에서 사용한 방법으로 받아 A배열에 집어넣었다. Visited 배열을 모두 false로 바꾸었으며, check배열을 모두 false로 두고 파일을 받아 해당하는 숫자를 가진 check 배열만 true로 바꾸는 기능을 구현하였다.

실행 결과 :

