

**มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ**

**KING MONGKUT’S UNIVERSITY OF TECHNOLOGY NORTH BANGKOK**

**ASSIGNMENT 1 LINKED LIST**

**เสนอ**

**อาจารย์ประดิษฐ์ พิทักษ์เสถียรกุล**

**จัดทำโดย**

**นายวรศิษฏ์ ภู่สุวรรณ์**

**ITI-2RB รหัส 6206021421237**

**รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithm**

**(รหัสรายวิชา 060223119)**

**ภาคการศึกษาที่ 1 ปีการศึกษา 2563**

**สาขา เทคโนโลยีสารสนเทศและการจัดการอุตสาหกรรม ภาควิชา เทคโนโลยีสารสนเทศ**

**คณะเทคโนโลยีและการจัดการอุตสาหกรรม**

**มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ ปราจีนบุรี**

**Code ไฟล์ LinkList.h**

template<class Object>

class List;

template<class Object>

class ListItr;

template<class Object>

class ListNode{

ListNode(const Object & theElement = Object(),ListNode \*n=NULL) : element(theElement),next(n){}

Object element;

ListNode \*next;

friend class List<Object>;

friend class ListItr<Object>;

}

template<class Object>

class ListItr{

public :

ListItr():current(NULL){}

bool isPastEnd() comst{return current == NULL;}

void advance(){

if(!isPastEnd()) current=current->next;

}

const Oject & retrieve() const{

if(isPastEnd()) throw BadIterator();

return current->element;

}

private :

ListNode<Object> \*current;

ListItr(ListNode<Object> \*current;) : current(theNode){}

friend class List<Object>;

}

template <class Object>

class List{

public :

List();

List(const List&rhs);

~List();

int FindKth((const List<Object> & theList,const a));

bool isEmpty() const;

void makeEmpty();

ListItr<Object> zeroth() const;

ListItr<Object> first() const;

void insert(const Object&x,const ListItr<Object>&p);

ListItr<Object> find (const Object&x) const;

ListItr<Object> findPrevious (const Object&x) const;

void remove(const Object&x);

const List&operator=(const List&rhs);

private:

ListNode<Object> \*header;

}

**Code ไฟล์ Listl.cpp**

#include<iostream>

#include <string>

using namespace std;

template<class Object>

class List;

template<class Object>

class ListItr;

template <class Object>

class ListNode

{

ListNode(const Object & theElement = Object(), ListNode \* n = NULL)

: element(theElement), next(n) {}

Object element;

ListNode \*next;

friend class List<Object>;

friend class ListItr<Object>;

};

template<class Object>

class ListItr

{

public:

ListItr() : current(NULL)

{

}

bool isPastEnd() const

{

return current == NULL;

}

void advance()

{

if (!isPastEnd())

current = current->next;

}

const Object & retrieve() const

{

if (isPastEnd())

return NULL;

return current->element;

}

private:

ListNode<Object> \*current;

ListItr(ListNode<Object> \*theNode) : current(theNode) { }

friend class List<Object>;

};

template<class Object>

class List

{

public:

List();

List(const List & rhs);

~List();

bool isEmpty() const;

void makeEmpty();

ListItr<Object> zeroth() const;

ListItr<Object> first() const;

void insert(const Object & x, const ListItr<Object> & p);

ListItr<Object> find(const Object & x) const;

ListItr<Object> findPrevious(const Object & x) const;

int remove(const Object & x);

const List & operator=(const List & rhs);

private:

ListNode<Object> \*header;

};

template <class Object>

List<Object>::List()

{

header = new ListNode<Object>;

}

template <class Object>

bool List<Object>::isEmpty() const

{

return header->next == NULL;

}

template <class Object>

ListItr<Object> List<Object>::zeroth() const

{

return ListItr<Object>(header);

}

template <class Object>

ListItr<Object> List<Object>::first() const

{

return ListItr<Object>(header->next);

}

template <class Object>

int printList(const List<Object> & theList)

{

int n = 0;

if (theList.isEmpty())

cout << "Empty List" << endl;

else

{

ListItr<Object> itr = theList.first();

for (; !itr.isPastEnd(); itr.advance()){

cout << "[" << itr.retrieve() << "] ";

n++;

}

}

cout << endl;

return n;

}

template<class Object>

ListItr<Object> List<Object>::find(const Object & x) const

{

ListNode<Object> \*itr = header->next;

while (itr != NULL && itr->element != x)

itr = itr->next;

cout << "ListNode Address is " << itr << endl;

return ListItr<Object>(itr);

}

template<class Object>

int List<Object>::remove(const Object & x)

{

ListItr<Object> p = findPrevious(x);

if (p.current->next != NULL)

{

ListNode<Object> \*oldNode = p.current->next;

p.current->next = p.current->next->next;

delete oldNode;

return 1;

}

return 0;

}

template <class Object>

ListItr<Object> List<Object>::findPrevious(const Object & x) const

{

ListNode<Object> \*itr = header;

while (itr->next != NULL && itr->next->element != x)

itr = itr->next;

return ListItr<Object>(itr);

}

template<class Object>

void List<Object>::insert(const Object & x, const ListItr<Object> & p)

{

if (p.current != NULL)

p.current->next = new ListNode<Object>(x, p.current->next);

}

template <class Object>

void List<Object>::makeEmpty()

{

while (!isEmpty())

remove(first().retrieve());

}

template<class Object>

List<Object>::~List()

{

makeEmpty();

delete header;

}

template<class Object>

void FindKth(const List<Object> & theList,int l)

{

ListItr<int> b;

b=theList.zeroth();

for(int i=0;i<=l;i++)

{

if (theList.isEmpty()){

cout << "Empty List" << endl;

i=l;

}

else

{ if(i!=l) b.advance();

else{

if(b.retrieve()==0) cout<<"Not found\n";

else cout<<" Element is "<<b.retrieve()<<"\n";

}

}

}

}

void main()

{

List<int> a;

ListItr<int> b;

ListItr<int> c;

int s,f=0,r;

char menu,menu2;

do

{

system("cls");

cout<<"==============================\n";

cout<<"| Main Menu |\n";

cout<<"==============================\n";

cout<<"| 1. Insert |\n";

cout<<"| 2. Print |\n";

cout<<"| 3. Find |\n";

cout<<"| 4. Remove |\n";

cout<<"| 5. FindKTh |\n";

cout<<"| 6. Quit |\n";

cout<<"==============================\n";

cout<<"Enter Choice : ";

cin>>menu;

switch(menu)

{

case '1' : do{

system("cls");

cout<<"-----------------------\n";

cout<<"| Menu |\n";

cout<<"-----------------------\n";

cout<<"| 1. Insert Front |\n";

cout<<"| 2. Insert Back |\n";

cout<<"| 3. Insert Position |\n";

cout<<"| 4. Quit |\n";

cout<<"-----------------------\n";

cout<<"Enter Choice : ";

cin>>menu2;

switch(menu2)

{

case '1':

b=a.zeroth();

cout << "Enter Element : ";

cin>>s;

a.insert(s,b);

system("pause");

break;

case '2':

b=a.findPrevious(NULL);

cout << "Enter Element : ";

cin>>s;

a.insert(s,b);

system("pause");

break;

case '3':

int m,p;

b=a.zeroth();

do{

m=printList(a);

cout << "Enter Position (not more " <<m +1<< ") : ";

cin >> p ;

if (p>m+1){cout << "No This Position Plase Try Again " << endl << endl;}

} while (p>m+1);

for (int i = 1; i<p; i++) b.advance();

cout << "Enter Element : ";

cin >> s;

a.insert(s, b);

system("pause");

break;

case '4': cout<<"Quit Program\n";

break;

default : cout<<"Don't have this menu!\n";

break;

}

}while(menu2!='4');

break;

case '2' :

int n;

n = printList(a);

cout << "Count of ListNode : " << n << endl << endl;

system("pause");

break;

case '3':

f = 0;

cout << "Enter Element : ";

cin >> s;

b = a.zeroth();

c = a.find(s);

if (c.retrieve() != NULL){

while (b.retrieve() != c.retrieve()){

b.advance();

f++;}

cout << "This Element is position " << f;

}

else

cout << "Find Not Found";

cout << endl << endl;

system("pause");

break;

case '4':

cout << "Enter Element : ";

cin >> s;

r = a.remove(s);

if (r == 1){

cout << "This Element is Removed" << endl;

}

else{

cout << "This Element is Not Remove" << endl;

}

system("pause");

break;

case '5':

cout<<"Enter Position : ";

cin>>s;

FindKth(a,s);

system("pause");

break;

case '6': cout<<"Quit Program\n";

break;

default : cout<<"Don't have this menu!\n";

system("pause");

break;

}

}

while(menu!='6');

}