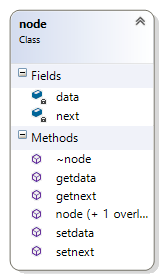
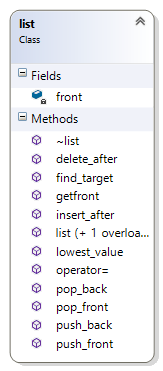
# Lab 9

Problem: Create a class that handles linked list processing

UML Diagrams:



## Source Files:

main.cpp:

#include "list.h"

int main()

{

list l1;

list l2;

int number;

cout<<"Filling list one with 0-10!";

for(int i=0;i<11;i++)

l1.push\_back(i);

cout<<"\nCopying list one to list two!";

l2=l1;

cout<<"\nCopying list one to a new list!";

list l3=l1;

cout<<"\n\nHere are the three lists!\nL1:"<<l1<<"\n\nL2:"<<l2<<"\n\nL3:"<<l3;

cout<<"\nWhat number to push to back of L1? ";

cin>>number;

l1.push\_back(number);

cout<<"\nWhat number would you like to push to front of L2? ";

cin>>number;

l2.push\_front(number);

cout<<"\nPopping front value from list one!";

l1.pop\_front();

cout<<"\nPopping back value from list two!";

l2.pop\_back();

cout<<"\nWhat number would you like to insert to list 1? ";

cin>>number;

int position;

cout<<"\nWhere would you like to insert the number? ";

cin>>position;

l1.insert\_after(number,position);

int target;

cout<<"\nWhat number would you like to find in list 1? ";

cin>>target;

cout<<"\nIt is in position "<<l2.find\_target(target);

cout<<"\nFinding the lowest value in list 2!";

cout<<"\nThe lowest value is: "<<l2.lowest\_value();

cout<<"\nWhere would you like to delete after in list 3? ";

cin>>position;

l3.delete\_after(position);

cout<<"\n\n\nHere are the lists, after changes!\nL1:"<<l1<<"\nL2:"<<l2<<"\nL3:"<<l3;

return 0;

}

list.h:

#pragma once

#include "node.h"

class list

{

public:

list(void);

list(const list& rhs);

~list(void);

private:

node\* front;

public:

void push\_front(int i);

node\* getfront(void)const;

friend ostream & operator<<(ostream & out,const list & second);

void pop\_front(void); //removes the first node, if there is one

void push\_back(int i);

void pop\_back(void);

void insert\_after(int i, int p);

int find\_target(int i);

list& operator=(const list & rhs);

int lowest\_value(void);

void delete\_after(int p);

};

list.cpp:

#include "list.h"

list::list(void)

{

front=0;

}

list::list(const list& rhs)

{

//front=rhs.getfront();

front=NULL;

for(node\* temp=rhs.getfront();temp!=NULL;temp=temp->getnext())

this->push\_back(temp->getdata());

}

list::~list(void)

{

cout<<"Destroying list\n";

if(front!=NULL)

delete front;

}

void list::push\_front(int i)

{

front=new node(i,front);

}

ostream & operator<<(ostream & out,const list & second)

{

node\* temp=second.getfront();

while(temp!=NULL)

{

out<<temp->getdata()<<" ";

temp=temp->getnext();

}

out<<endl;

return out;

}

node\* list::getfront(void)const

{

return front;

}

void list::pop\_front(void) //removes the first node, if there is one

{

if(front!=NULL)

{

node\* temp=front;

front=front->getnext();

temp->setnext(NULL);

delete temp;

}

}

void list::push\_back(int i)

{

if(front==NULL)

front=new node(i,NULL);

else if(front->getnext()==NULL)

front->setnext(new node(i,NULL));

else

{

node\* temp=front;

while(temp->getnext()!=NULL)

{

temp=temp->getnext();

}

temp->setnext(new node(i,NULL));

}

}

void list::pop\_back(void)

{

if(front->getnext()==NULL)

delete front;

if(front!=NULL)

{

node\* temp=front;

node\* temp2=front;

while(temp->getnext()!=NULL)

{

temp=temp->getnext();

if(temp->getnext()==NULL)

{

temp2->setnext(NULL);

break;

}

temp2=temp2->getnext();

}

delete temp;

}

}

void list::insert\_after(int i, int p)

{

node\* temp=front;

while(temp->getnext()!=NULL)

{

if(p==0)

{

node\* temp2=new node(i,temp->getnext());

temp->setnext(temp2);

}

temp=temp->getnext();

p--;

}

}

int list::find\_target(int i)

{

node\* temp=front;

int a=0;

while(temp->getnext()!=NULL)

{

if(temp->getdata()==i)

return a;

a++;

temp=temp->getnext();

}

return -1;

}

list& list::operator=(const list & rhs)

{

delete front;

front=NULL;

node\* temp=rhs.getfront();

while(temp!=NULL)

{

this->push\_back(temp->getdata());

temp=temp->getnext();

}

return \*this;

}

int list::lowest\_value(void)

{

//assume list is not empty

int low=front->getdata();

for(node\* ptr=front;ptr!=NULL;ptr=ptr->getnext())

{

if(ptr->getdata()<low)

low=ptr->getdata();

}

return low;

}

void list::delete\_after(int p)

{

node\* temp=getfront();

while(temp->getnext()!=NULL)

{

temp=temp->getnext();

p--;

if(p==0)

{

node\* temp2=temp;

temp2=temp2->getnext();

temp->setnext(temp2->getnext());

temp2->setnext(NULL);

delete temp2;

}

}

}

node.h:

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

class node

{

public:

node(void); //default constructor

node(int d, node\* n); //initializing constructor

~node(void);

private:

int data;

node\* next;

public:

void setdata(int i);

void setnext(node\* ptr);

int getdata(void);

node\* getnext(void);

};

node.cpp:

#include "node.h"

node::node(void)

: data(0)

, next(NULL)

{

}

node::~node(void)

{

if(next!=NULL)

delete next;

cout<<"destroying node with data = "<<data<<endl;

}

node::node(int d, node\* n)

:data(d),next(n)

{

}

void node::setdata(int i)

{

data=i;

}

void node::setnext(node\* ptr)

{

next=ptr;

}

int node::getdata(void)

{

return data;

}

node\* node::getnext(void)

{

return next;

}

## Sample Run:

Filling list one with 0-10!

Copying list one to list two!

Copying list one to a new list!

Here are the three lists!

L1:0 1 2 3 4 5 6 7 8 9 10

L2:0 1 2 3 4 5 6 7 8 9 10

L3:0 1 2 3 4 5 6 7 8 9 10

What number to push to back of L1? 21

What number would you like to push to front of L2? 55

Popping front value from list one!destroying node with data = 0

Popping back value from list two!destroying node with data = 10

What number would you like to insert to list 1? 6

Where would you like to insert the number? 7

What number would you like to find in list 1? 8

It is in position 9

Finding the lowest value in list 2!

The lowest value is: 0

Where would you like to delete after in list 3? 9

destroying node with data = 10

Here are the lists, after changes!

L1:1 2 3 4 5 6 7 8 6 9 10 21

L2:55 0 1 2 3 4 5 6 7 8 9

L3:0 1 2 3 4 5 6 7 8 9

Destroying list

destroying node with data = 9

destroying node with data = 8

destroying node with data = 7

destroying node with data = 6

destroying node with data = 5

destroying node with data = 4

destroying node with data = 3

destroying node with data = 2

destroying node with data = 1

destroying node with data = 0

Destroying list

destroying node with data = 9

destroying node with data = 8

destroying node with data = 7

destroying node with data = 6

destroying node with data = 5

destroying node with data = 4

destroying node with data = 3

destroying node with data = 2

destroying node with data = 1

destroying node with data = 0

destroying node with data = 55

Destroying list

destroying node with data = 21

destroying node with data = 10

destroying node with data = 9

destroying node with data = 6

destroying node with data = 8

destroying node with data = 7

destroying node with data = 6

destroying node with data = 5

destroying node with data = 4

destroying node with data = 3

destroying node with data = 2

destroying node with data = 1

Press any key to continue . . .