Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа**

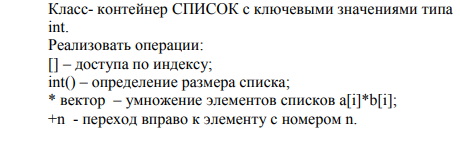
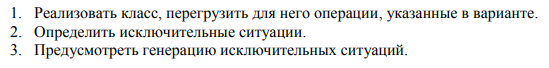
**"Классы №9”**

Выполнил:   
студент группы РИС-23-1б   
Агзамов Артур Альферович

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

2024 г.

**Задача:**



**Анализ задачи:**

**Код:**

#include"class-9-error.h"

#include"class-9.h"

#include<iostream>

#include<string>

using namespace std;

int main(){

system("chcp 1251>null");

system("cls");

system("chcp 1251");

system("cls");

List sp1,sp2;

int count,x,index,h;

cout << "Enter element count: ";

cin >> count;

try

{

if (count < 1) throw error\_empty();

cout << "Enter element of list: ";

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

{

cin >> x;

sp1.push\_back(x);

}

cout << "Enter index of list number 1 for calling: ";

cin >> index;

cout << "Enter element with this index: " << sp1[index] << endl;

cout << "Enter index of right element of list sp1: ";

cin >> h;

cout << "element with this index: " << sp1+h << endl;

cout << "Size of lsit : " << sp1() << endl;

cout << "Enter count of element of list number 2: ";

cin >> count;

if (count < 1) throw error\_empty();

cout << "Enter element of list: ";

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

{

cin >> x;

sp2.push\_back(x);

}

sp1\* sp2;

cout << "List obtained by multiplying two lists: ";

cout<<sp1<<endl;

}

catch (Error& a)

{

a.what();

}

return 0;

}  
#pragma once

#include<string>

#include<iostream>

using namespace std;

class Error{

protected:

string msg;

public:

virtual void what(){};

};

class IndexError:public Error{

protected:

string msg;

public:

IndexError(){msg="Index Error\n";}

virtual void what(){cout<<msg;}

};

class Index\_min\_error:public IndexError{

protected:

string msg\_;

public:

Index\_min\_error(){IndexError();msg\_="Index<MinIndex\n";}

virtual void what(){cout<<msg<<msg\_;}

};

class Index\_max\_error:public IndexError{

protected:

string msg\_;

public:

Index\_max\_error(){IndexError();msg\_="Index>MaxIndex\n";}

virtual void what(){cout<<msg<<msg\_;}

};

class size\_error:public Error{

protected:

string msg;

public:

size\_error(){msg="Index Error\n";}

virtual void what(){cout<<msg;}

};

class error\_empty :public size\_error

{

protected:

string msg\_;

public:

error\_empty(){size\_error();msg\_ = "empty list";}

virtual void what(){cout<<msg<<msg\_;}

};

class error\_diff\_size :public size\_error

{

protected:

string msg\_;

public:

error\_diff\_size(){size\_error();msg\_= "Diff size of list";}

virtual void what(){cout<<msg<<msg\_;}

};  
#pragma once

#include <iostream>

#include <vector>

using namespace std;

struct Node {

public:

int data;

Node\* next;

Node\* prev;

};

class List {

private:

int size;

Node\* head;

Node\* tail;

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

friend istream& operator >> (istream& in, const List& list);

public:

List() : size(0), head(nullptr),tail(nullptr) {}

void push\_back(int data){

Node\* new\_node = new Node;

new\_node->data=data;

new\_node->next=nullptr;

if(this->head==nullptr)

{

this->head=new\_node;

this->tail=new\_node;

}

else

{

tail->next = new\_node;

new\_node->prev = tail;

tail=new\_node;

}

this->size++;

}

void push\_front(int data){

Node\* new\_node = new Node;

new\_node->data=data;

if(this->head==nullptr)

{

this->head=new\_node;

this->tail=new\_node;

this->size++;

}

else

{

head->prev = new\_node;

new\_node->next = head;

head=new\_node;

this->size++;

}

}

int front(){

return this->head->data;

}

int back(){

return this->tail->data;

}

int pop\_back(){

int temp;

if(this->tail!=nullptr){

Node \* current\_node= this->tail;

tail=current\_node->prev;

temp=current\_node->data;

tail->next=nullptr;

this->size--;

}

return temp;

}

int pop\_front(){

int temp;

if(this->head!=nullptr){

Node \* current\_node= this->head;

head=current\_node->next;

temp=current\_node->data;

head->prev=nullptr;

this->size--;

}

return temp;

}

bool is\_empty(){

return this->size==0;

}

List& operator = (const List& list){

cout<<"Operator ="<<endl;

if(this == &list){

return \*this;

}

while (head!=nullptr){

Node\* temp=head;

head=head->next;

delete temp;

}

size =0;

Node\* current\_node=list.head;

while(current\_node!=nullptr){

push\_back(current\_node->data);

current\_node=current\_node->next;

}

return \*this;

}

int& operator [](int index){

if (index < 0) throw Index\_min\_error();

if (index >= size) throw Index\_max\_error();

if(index <this->size&&index>=0){

Node\* current\_node = this->head;

for(int i=0;i!=index;i++){

current\_node=current\_node->next;

}

return current\_node->data;

}

else{

cerr<<"index out of range";

exit(0);

}

}

int operator +(int h)

{

if (h < 0) throw Index\_min\_error();

if (h >= size) throw Index\_max\_error();

Node\* c = head;

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

c = c->next;

return (c->data);

}

int operator () (){

if(size==0)throw error\_empty();

return this->size;

}

List operator \* (List& list){

if (this->size != list.size)

{

throw error\_diff\_size();

}

int temp\_size;

if(this->size>list.size){

temp\_size=list.size;

}

else

{

temp\_size=this->size;

}

List temp(temp\_size,0);

for(int i=0;i<temp\_size;i++){

temp[i]=(\*this)[i]\*list[i];

}

return temp;

}

List(int size,int data){

this->size=size;

if(size>0){

Node\* node = new Node;

node->data=data;

this->head=node;

this->tail=node;

for(int i=1;i<size; i++){

Node\* newNode = new Node;

newNode->data=data;

tail->next = newNode;

newNode->prev = tail;

tail=newNode;

}

tail->next=nullptr;

}

else

{

this->head=nullptr;

this->tail=nullptr;

}

}

List(const List& list)

{

this->head=nullptr;

this->tail=nullptr;

this->size=0;

Node\* current\_node=list.head;

while(current\_node!=nullptr){

push\_back(current\_node->data);

current\_node=current\_node->next;

}

};

List(int size){

this->size=size;

if(size>0){

Node\* node = new Node;

this->head=node;

this->tail=node;

for(int i=1;i<size; i++){

Node\* newNode = new Node;

tail->next = newNode;

newNode->prev = tail;

tail=newNode;

}

tail->next=nullptr;

}

else

{

this->head=nullptr;

this->tail=nullptr;

}

};

~List()

{

Node\* current\_node=head;

while(current\_node!=nullptr){

Node\* next=current\_node->next;

delete current\_node;

current\_node=next;

}

head=nullptr;

};

};

ostream& operator<<(ostream& out, const List& list){

out <<endl<< "Elements of list"<<endl;

Node\* current\_node = list.head;

while (current\_node!=nullptr)

{

out<<current\_node->data<< " ";

current\_node=current\_node->next;

}

out<<endl<<"conclusion end"<<endl;

return out;

}

istream& operator >> (istream& in, const List& list){

cout<<endl<<"Enter element of list"<<endl;

Node\* current\_node=list.head;

while (current\_node!=nullptr){

in>>current\_node->data;

current\_node=current\_node->next;

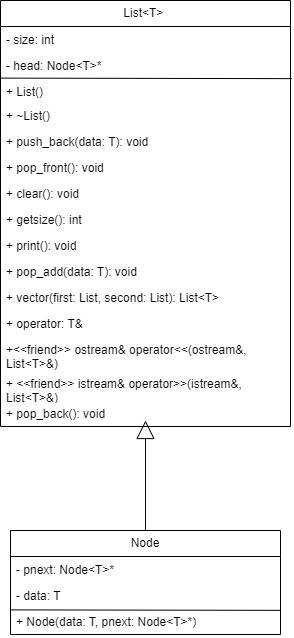
}

cout<<endl<<"Enter element stop"<<endl;

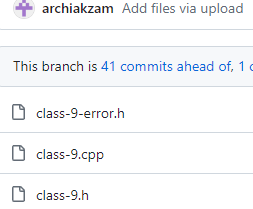
return in;

}

**UML диаграмма:**



Скрины из git:



**Выводы:** программа сработала корректно и вывела все возможные решения.