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

ПНИПУ

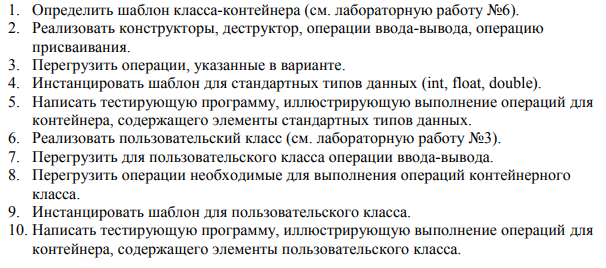
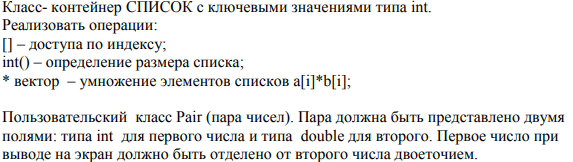
**Лабораторная работа**

**"Классы №7”**

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

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

2024 г.

**Задача:**  


**Анализ задачи:**

**Код:**

#include <iostream>

#include <vector>

#include "class-7.h"

using namespace std;

int main() {

Pair pair1(2,5.4);

Pair pair2(4,5.1);

List list1(2,pair1);

list1.push\_front(pair2);

cout<<list1<<endl;

return 0;

}

#pragma once

#include <iostream>

#include <vector>

using namespace std;

class Pair {

private:

int first;

double second;

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

public:

Pair() : first(0), second(0.0) {}

Pair(int f, double s) : first(f), second(s) {}

Pair operator\*(const Pair& p) const {

Pair result;

result.first = this->first \* p.first;

result.second = this->second \* p.second;

return result;

}

friend ostream& operator<<(ostream& out, const Pair& p) {

out << p.first << " : " << p.second;

return out;

}

};

struct Node {

public:

Pair 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:

void push\_back(const Pair& 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(const Pair& 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++;

}

}

Pair front(){

return this->head->data;

}

Pair back(){

return this->tail->data;

}

Pair pop\_back(){

Pair temp;

if(this->tail != nullptr) {

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

tail = current\_node->prev;

temp = current\_node->data;

tail->next = nullptr;

delete current\_node;

this->size--;

}

return temp;

}

Pair pop\_front(){

Pair 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;

}

Pair& operator [](int index){

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 () (){

return this->size;

}

List operator \* (List& list){

int temp\_size;

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

temp\_size=list.size;

}

else

{

temp\_size=this->size;

}

List temp(temp\_size,Pair(0, 0));

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

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

}

return temp;

}

List(int size, const Pair& data) {

this->size = size;

if (size > 0) {

head = new Node;

head->data = data;

tail = head;

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 {

head = nullptr;

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;

};

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

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

Node\* current = list.head;

while (current != nullptr) {

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

current = current->next;

}

out << endl << "End of list" << endl;

return out;

}

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

Node\* current = list.head;

cout << "Enter elements of the list:" << endl;

while (current != nullptr) {

in >> current->data.first >> current->data.second;

current = current->next;

}

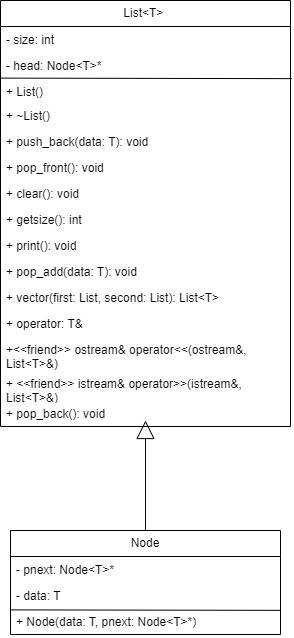
cout << "End of input" << endl;

return in;

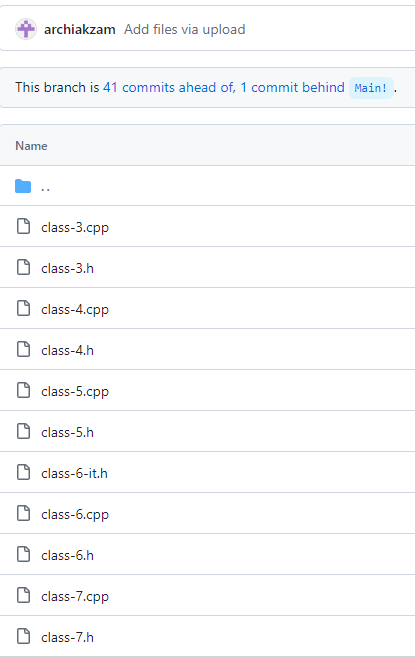
}

};

**UML диаграмма:**



Скрины из git:



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