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

ПНИПУ

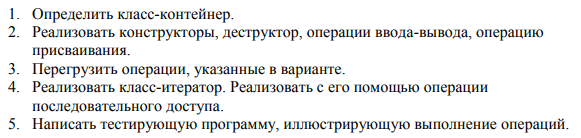
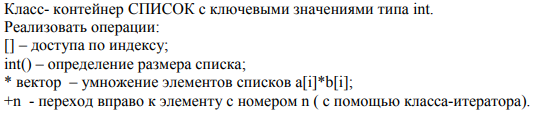
**Лабораторная работа**

**"Классы №6”**

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

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

2024 г.

**Задача:**  
  


**Анализ задачи:**

1. Определение класса-контейнера Список со значениями типа int в качестве ключей.

2. Реализация конструкторов, деструктора, операций ввода-вывода и операции присваивания для класса Список.

3. Перегрузка операций доступа по индексу (), определения размера списка (int()), умножения элементов списков (вектор) и операции перехода к элементу по индексу с помощью класса-итератора (+n).

4. Написание класса-итератора, который позволит реализовать операции последовательного доступа к элементам списка.

5. Разработка тестирующей программы для иллюстрации выполнения всех перегруженных операций и использования класса-итератора.

6. Тестирование программы для проверки корректности выполнения операций доступа по индексу, определения размера списка, умножения элементов списков и перемещения к элементам с помощью класса-итератора.

**Код:**

#include <iostream>

#include <vector>

#include "class-6.h"

using namespace std;

int main() {

List list1(10,0);

cout<<list1<<endl;

return 0;

}

#pragma once

#include "class-6-it.h"

#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);

Iterator begin;

Iterator end;

public:

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;

this->begin.elem=this->head;

this->end.elem=this->tail;

}

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++;

this->begin.elem=this->head;

this->end.elem=this->tail;

}

else

{

head->prev = new\_node;

new\_node->next = head;

head=new\_node;

this->size++;

this->begin.elem=this->head;

}

}

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;

}

this->begin=list.begin;

this->end=list.end;

return \*this;

}

int& 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,0);

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

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

}

return temp;

}

Iterator first(){

return this->begin;

}

Iterator last(){

return this->end;

}

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;

}

this->begin.elem=this->head;

this->end.elem=this->tail;

}

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;

}

this->begin.elem=this->head;

this->end.elem=this->tail;

};

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;

}

this->begin.elem=this->head;

this->end.elem=this->tail;

};

~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;

}

class Iterator

{

private:

friend class List;

Node\* elem;

public:

Iterator()

{

this->elem=nullptr;

}

Iterator(const Iterator& iterator)

{

this->elem=iterator.elem;

}

Iterator& operator = (const Iterator& iterator)

{

this->elem=iterator.elem;

return \*this;

}

bool operator ==( const Iterator& iterator)

{

return this->elem==iterator.elem;

}

bool operator !=( const Iterator& iterator)

{

return this->elem!=iterator.elem;

}

Iterator& operator++(){

this-> elem= this->elem->next;

return \*this;

}

Iterator& operator--(){

this-> elem= this->elem->prev;

return \*this;

}

Iterator& operator+(const int number)const{

Iterator temp(\*this);

for(int i=0;i<number;i++){

temp.elem= temp.elem->next;

}

return temp;

}

Iterator& operator-(const int number)const{

Iterator temp(\*this);

for(int i=0;i<number;i++){

temp.elem= temp.elem->prev;

}

return temp;

}

int& operator\* ()

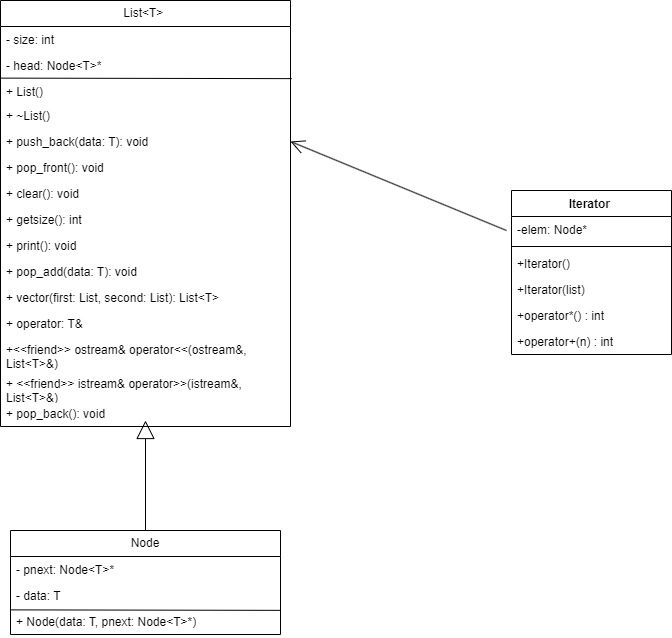
{

return this->elem->data;

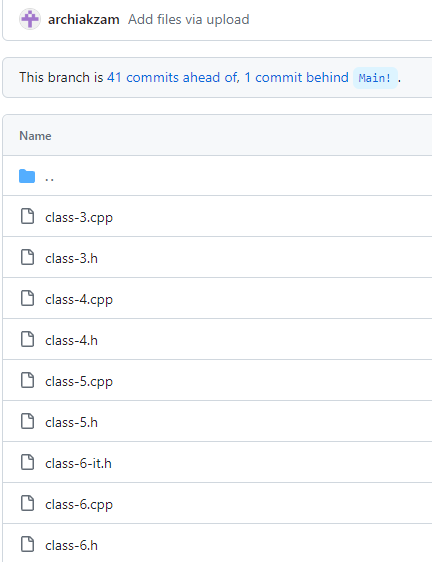
}

};

**UML диаграмма:**



Скрины из git:



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