Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет», ПНИПУ

ЛАБОРАТОРНАЯ РАБОТА

ШАБЛОНЫ КЛАССОВ

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

**Постановка задачи**

1. Определить шаблон класса-контейнера.
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Инстанцировать шаблон для стандартных типов данных (int, float, double).
5. Реализовать пользовательский класс.
6. Перегрузить операции ввода-вывода.
7. Перегрузить операции необходимые для выполнения операций контейнерного класса.
8. Инстанцировать шаблон для пользовательского класса.

Для варианта 15:

Класс – контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции:

[] – доступ про индексу;

int() – определение размера списка;

\* вектор – умножение элементов списков a[i] \* b[i];

+n переход вправо к элементу с номером n (с помощью класса-итератора)

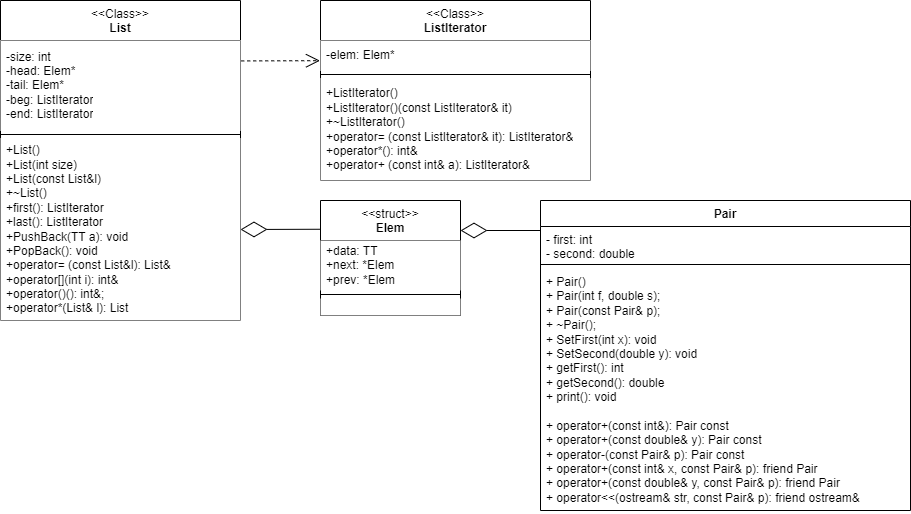
Создать класс Pair (пара чисел). Пара должна быть представлена двумя полями: первое – int, второе – double. При выводе на экран первое число должно быть отделено от второго двоеточием. Реализовать:

* Вычитание пар чисел
* Добавление константы к паре (если константа типа int, то прибавление к первому полю, если double – ко второму)

**Анализ задачи**

1. Определение и описание класса и его методов производится в одном файле из-за специфики работы шаблонов.
2. Для корректного ввода и вывода значений пользовательского класса перегружаем операции потокового ввода-вывода.
3. Проверка будет производиться на списках содержащих значения типа double и Pair.

**UML**



**Код**

Pair.h

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

Pair();

Pair(int f, double s);

Pair(const Pair& p);

~Pair();

void SetFirst(int x);

void SetSecond(double y);

int getFirst();

double getSecond();

void print();

//перегруженные операции

Pair operator+(const int& x) const;

Pair operator+(const double& y) const;

Pair operator-(const Pair& p) const;

friend Pair operator+(const int& x, const Pair& p);

friend Pair operator+(const double& y, const Pair& p);

friend ostream& operator<<(ostream& str, const Pair& p);

friend istream& operator>>(istream& str, Pair& p);

};

Pair.cpp

#pragma once

#include "Pair.h"

//перегрузка операции присваивания

Pair::Pair() {

first = 0;

second = 0;

}

Pair::Pair(int f = 0, double s = 0) {

this->first = f;

this->second = s;

}

Pair::Pair(const Pair& p) {

this->first = p.first;

this->second = p.second;

}

Pair::~Pair() {

}

int Pair::getFirst() { return this->first; }

void Pair::SetFirst(int x) { this->first = x; }

double Pair::getSecond() { return this->second; }

void Pair::SetSecond(double y) { this->second = y; }

void Pair::print() {

cout << this->first << " : " << this->second << endl;

}

//перегруженные операции

Pair Pair::operator+(const int& x) const {

return Pair(this->first + x, this->second);

}

Pair Pair::operator+(const double& y) const {

return Pair(this->first, this->second + y);

}

Pair Pair::operator-(const Pair& p) const {

return Pair(this->first - p.first, this->second - p.second);

}

//Дружественные перегруженные операции

Pair operator+(const int& x, const Pair& p) {

return p + x;

}

Pair operator+(const double& y, const Pair& p) {

return p + y;

}

ostream& operator<<(ostream& str, const Pair& p) {

str << " " << p.first << " : " << p.second << endl;

return str;

}

istream& operator>>(istream& str, Pair& p) {

cout << "Enter int value: ";

str >> p.first;

cout << "Enter double value: ";

str >> p.second;

return str;

}

List.h

#pragma once

const int MAX\_SIZE = 25;

template <typename TT>

class Elem{

public:

TT data;

Elem\* next = nullptr;

Elem\* prev = nullptr;

};

template <typename TT>

class ListIterator

{

public:

Elem<TT>\* elem;

ListIterator();

ListIterator(const ListIterator<TT>& it);

ListIterator& operator=(const ListIterator<TT>& a);

TT& operator\*() const;

ListIterator& operator+ (const int& a);

};

template <typename TT>

class List

{

int size;

Elem<TT>\* head = nullptr;

Elem<TT>\* tail = nullptr;

ListIterator<TT> beg;

ListIterator<TT> end;

public:

List() { size = 0; head = nullptr; tail = nullptr;};

List(int s);

List(const List<TT>& l);

~List();

ListIterator<TT> first();

ListIterator<TT> last();

void PushBack(TT a);

void PopBack();

List& operator= (const List<TT>& l);

TT& operator[] (int i);

int& operator()();

List operator\*(List<TT>& l);

};

template <typename TT>

List<TT>::List(int s)

{

size = s;

Elem<TT>\* new\_elem = new Elem<TT>;

head = new\_elem;

tail = new\_elem;

for (int i = 0; i < size - 1; i++)

{

Elem<TT>\* new\_elem = new Elem<TT>;

tail->next = new\_elem;

new\_elem->prev = tail;

tail = new\_elem;

}

beg.elem = head;

end.elem = tail;

}

template <typename TT>

List<TT>::List(const List<TT>& l)

{

Elem<TT>\* elem = l.head;

while (elem != nullptr) {

PushBack(elem->data);

elem = elem->next;

}

beg.elem = head;

end.elem = tail;

}

template <typename TT>

List<TT>::~List()

{

Elem<TT>\* curr = head;

while (curr != nullptr)

{

head = curr->next;

delete curr;

curr = head;

}

}

template <typename TT>

List<TT>& List<TT>::operator= (const List<TT>& l)

{

if (this == &l)return \*this;

if (head != nullptr)

{

Elem<TT>\* curr = head;

while (curr != nullptr)

{

head = curr->next;

delete curr;

curr = head;

}

}

Elem<TT>\* elem = l.head;

while (elem != nullptr) {

PushBack(elem->data);

elem = elem->next;

}

beg.elem = head;

end.elem = tail;

return \*this;

}

template <typename TT>

TT& List<TT>::operator[] (int i)

{

Elem<TT>\* curr = head;

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

curr = curr->next;

}

return curr->data;

}

template <typename TT>

List<TT> List<TT>::operator\*(List<TT>& l)

{

List base = \*this;

int s = this->size;

List temp;

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

{

temp.PushBack(base[i] \* l[i]);

}

return temp;

}

template <typename TT>

int& List<TT>::operator()() {

return size;

}

template <typename TT>

ListIterator<TT> List<TT>::first()

{

return beg;

}

template <typename TT>

ListIterator<TT> List<TT>::last()

{

return end;

}

template <typename TT>

void List<TT>::PushBack(TT a)

{

Elem<TT>\* new\_elem = new Elem<TT>;

new\_elem->data = a;

if (size == 0)

{

head = new\_elem;

tail = new\_elem;

size++;

beg.elem = head;

end.elem = tail;

}

else

{

tail->next = new\_elem;

new\_elem->prev = tail;

tail = new\_elem;

size++;

end.elem = tail;

}

}

template <typename TT>

void List<TT>::PopBack()

{

Elem<TT>\* curr = tail;

tail = curr->prev;

delete curr;

tail->next = nullptr;

size--;

end.elem = tail;

}

template <typename TT>

ListIterator<TT>::ListIterator() {

elem = nullptr;

}

template <typename TT>

ListIterator<TT>::ListIterator(const ListIterator& it) {

elem = it.elem;

}

template <typename TT>

ListIterator<TT>& ListIterator<TT>::operator=(const ListIterator<TT>& a)

{

elem = a.elem;

return \*this;

}

template <typename TT>

TT& ListIterator<TT>::operator\*() const

{

return elem->data;

}

template <typename TT>

ListIterator<TT>& ListIterator<TT>::operator+ (const int& a)

{

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

elem = elem->next;

}

return \*this;

}

Main.cpp

#include "List.h"

#include "Pair.h"

#include <iostream>

using namespace std;

int main()

{

int count, index;

double in\_value;

List<double> list1;

ListIterator <double> iter;

cout << "Enter number of elements: ";

cin >> count;

cout << endl;

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

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

{

cin >> in\_value;

list1.PushBack(in\_value);

}

cout << endl;

iter = list1.first();

cout << "Enter index of element in list to call: ";

cin >> index;

cout << list1[index] << endl;

cout << endl;

cout << "Enter number of elements to move for from first: ";

cin >> index;

iter + index;

cout << \*iter << endl;

cout << endl;

List<Pair> list2;

Pair new\_in\_value;

cout << "Enter number of elements of new list: ";

cin >> count;

cout << endl;

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

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

{

cin >> new\_in\_value;

list2.PushBack(new\_in\_value);

}

cout << "Enter index of element in list to call: ";

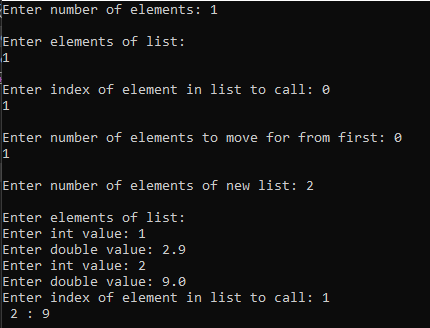
cin >> index;

cout << list2[index] << endl;

cout << endl;

}

**Решение**



**Выводы**

В ходе выполнения работы были изучены механизмы создания шаблонных классов и работы с ними.

**Ответы на контрольные вопросы:**

1. Шаблоны позволяют классам работать с разными типами данных.
2. Перед определением функции конструкция template <typename /имя шаблона/>, после определение функции.
3. Перед определением класса конструкция template <typename /имя параметра шаблона/>, после определение класса. Определение методов необходимо проводить в том же файле, где и определение класса.
4. Тип данных.
5. Может быть любым инстанцированным типом данных, может быть несколько параметров, функция может содержать не параметризированные формальные параметры.
6. template <typename /имя параметра шаблона/>
7. Да.
8. Определение класса и методов производится в одном файле, конкретные экземпляры формируются по ходу выполнения программы.
9. Да.
10. Нет.
11. Нет.
12. С использованием конструкции template <> перед определением.
13. Создание конкретного экземпляра в соответствии с указанным типом.
14. Во время создания экземпляра класса.

**Github**

<https://github.com/Hitikov/Lab_OOP_7>