Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

# Лабораторная работа №7

# «Шаблоны классов»

# Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Юхновец В.Г.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

1. Определить шаблон класса-контейнера (см. лабораторную работу №6).

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. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы стандартных типов данных.

6. Реализовать пользовательский класс (см. лабораторную работу №3).

7. Перегрузить для пользовательского класса операции ввода-вывода.

8. Перегрузить операции необходимые для выполнения операций контейнерного класса.

9. Инстанцировать шаблон для пользовательского класса.

10. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского

**Вариант 15**

Класс - контейнер СПИСОК с ключевыми значениями типа int.   
Реализовать операции:   
[] – доступа по индексу;   
int() – определение размера списка;   
\* вектор – умножение элементов списков a[i]\*b[i];   
Пользовательский класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием.

**Ответы на контрольные вопросы**

1. **В чем смысл использования шаблонов?**Ответ: Шаблоны в C++ позволяют создавать обобщенные функции и классы, которые могут работать с различными типами данных и значениями.
2. **Каковы синтаксис/семантика шаблонов функций?**Ответ: Шаблоны функций в C++ описываются с использованием ключевого слова "template" и параметров шаблона в угловых скобках.   
   Пример:

template<typename T>

T max(T a, T b) {

return (a > b) ? a : b;

}

1. **Тело шаблонной функции содержит код, который работает с параметрами шаблона. Каковы синтаксис/семантика шаблонов классов?**   
   Ответ: Шаблоны классов в C++ описываются с использованием ключевого слова "template" и параметров шаблона в угловых скобках.

Пример:

template<typename T>

class Array {

public:

void insert(T element);

T get(int index);

private:

T\* data;

int size;

};

1. **Тело шаблона класса содержит определение класса, которое может использовать параметры шаблона. Что такое параметры шаблона функции?**   
   Ответ: Параметры шаблона функции - это типы данных или значения, которые передаются в шаблонную функцию как аргументы при ее вызове. Они используются для создания обобщенных функций, которые могут работать с различными типами данных и значениями.
2. **Перечислите основные свойства параметров шаблона функции.**Ответ: Основные свойства параметров шаблона функции включают их типы данных, значения по умолчанию, их количество и порядок, а также их имена.
3. **Как записывать параметр шаблона?**   
   Ответ: Параметр шаблона записывается в угловых скобках после ключевого слова "template".  
   Пример:   
   template<typename T> void foo(T arg);
4. **Можно ли перегружать параметризованные функции?**Ответ: Да, параметризованные функции могут быть перегружены как и обычные функции, но с различными параметрами шаблона.
5. **Перечислите основные свойства параметризованных классов.**   
   Ответ: Основные свойства параметризованных классов включают их типы данных, значения по умолчанию, их количество и порядок, а также их имена.
6. **Все ли компонентные функции параметризованного класса являются параметризованными?**  
   Ответ: Нет, не все компонентные функции параметризованного класса обязательно должны быть параметризованными. Компонентные функции могут использовать параметры шаблона класса, но могут также иметь обычные параметры
7. **Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?**Ответ: Да, дружественные функции, описанные в параметризованном классе, также должны быть параметризованными, если они используют параметры шаблона класса.
8. **Могут ли шаблоны классов содержать виртуальные компонентные функции?**Ответ: Да, шаблоны классов могут содержать виртуальные компонентные функции, которые могут быть переопределены в производных классах.
9. **Как определяются компонентные функции параметризованных классов вне определения шаблона класса?**Ответ: Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса, как и обычные функции, но с использованием синтаксиса шаблонных функций.
10. **Что такое инстанцирование шаблона?**Ответ: Инстанцирование шаблона - это процесс создания экземпляра функции или класса на основе шаблона, путем подстановки конкретных типов данных или значений параметров шаблона.
11. **На каком этапе происходит генерирование определения класса по шаблону?**   
    Ответ: Генерирование определения класса по шаблону происходит на этапе компиляции, когда компилятор создает инстанцию шаблона, подставляя конкретные типы данных или значения параметров шаблона, и генерирует определение класса соответствующим образом.

**Диаграмма класса**

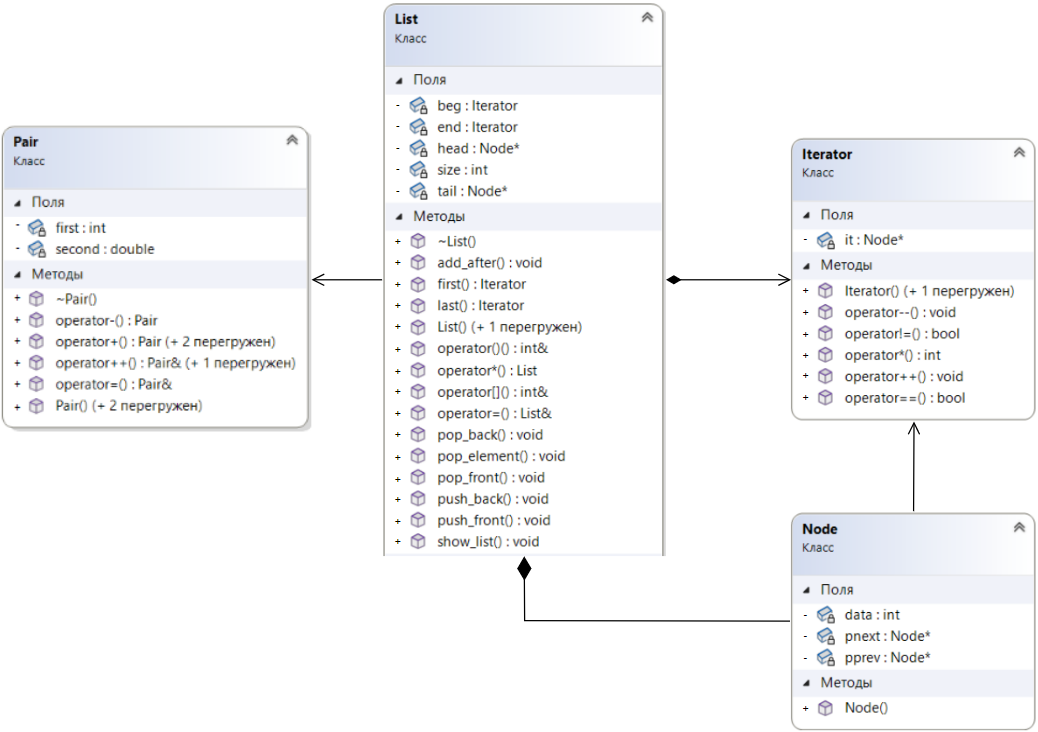


Рисунок 1 – классы List, Iterator, Pair и Node

**Описание классов**

Class List со вложенными классами Node и Iterator

#pragma once

#include <iostream>

#include <string>

using namespace std;

class List

{

private:

class Node

{

friend class List;

private:

int data;

Node\* pnext = nullptr;

Node\* pprev = nullptr;

public:

Node(int data);

};

int size = 0;

Node\* head = nullptr;

Node\* tail = nullptr;

class Iterator

{

friend class List;

private:

Node\* it = nullptr;

public:

Iterator();

Iterator(const Iterator& it);

bool operator==(const Iterator& it);

bool operator!=(const Iterator& it);

void operator++();

void operator--();

int operator\*();

};

Iterator beg;

Iterator end;

public:

List();

List(List& object);

~List();

void push\_front(int data);

void pop\_front();

void push\_back(int data);

void pop\_back();

void show\_list();

void add\_after(int number, int data);

void pop\_element(int number);

int& operator[](int index) const;

int& operator()();

List& operator=(const List& object);

List operator\*(List& object);

Iterator first()

{

if (head != nullptr)

beg.it = head;

return beg;

}

Iterator last()

{

if (tail != nullptr)

end.it = tail->pnext;

return end;

}

};

Class Pair

#pragma once

#include <iostream>

#include <iomanip>

using namespace std;

class Pair

{

friend ostream& operator<<(ostream& out, const Pair& ob);

friend istream& operator>>(istream& in, Pair& ob);

private:

int first;

double second;

public:

Pair();

Pair(int first, double second);

Pair(const Pair& ob);

~Pair();

Pair operator-(Pair& pair) const;

Pair operator+(Pair& pair) const;

Pair operator+(const int data) const;

Pair operator+(const double data) const;

Pair& operator++();

Pair operator++(int);

Pair operator\*(Pair& pair);

Pair& operator=(const Pair& pair);

};

**Определение методов**

Class List и вложенный Node

#include "List.h"

List::List()

{

size = 0;

head = nullptr;

tail = nullptr;

}

List::List(List& object)

{

for (int i = 0; i < object.size; i++)

(\*this).push\_back(object[i]);

}

List::~List()

{

if (head != nullptr)

{

Node\* current = head;

while (current != nullptr)

{

Node\* next = current->pnext;

delete current;

current = next;

}

head = nullptr;

tail = nullptr;

size = 0;

}

}

List::Node::Node(int data)

{

this->data = data;

}

void List::push\_front(int data)

{

if (head == nullptr)

{

head = new Node(data);

tail = head;

}

else

{

Node\* temp = head;

head = new Node(data);

temp->pprev = head;

head->pnext = temp;

}

++size;

}

void List::pop\_front()

{

if (head != nullptr)

{

Node\* temp = head->pnext;

delete head;

head = temp;

if (head != nullptr)

head->pprev = 0;

else

tail = nullptr;

--size;

}

else

{

system("cls"); cout << "Список пуст!\n\n";

}

}

void List::push\_back(int data)

{

if (head == nullptr)

{

head = new Node(data);

tail = head;

}

else

{

Node\* current = tail;

current->pnext = new Node(data);

tail = current->pnext;

tail->pprev = current;

tail->pnext = 0;

}

++size;

}

void List::pop\_back()

{

if (head != nullptr)

{

Node\* current = tail->pprev;

delete tail;

tail = current;

if (tail != nullptr)

current->pnext = nullptr;

else

head = nullptr;

--size;

}

else { system("cls"); cout << "Список пуст!\n\n"; }

}

void List::show\_list()

{

/\*system("cls");\*/

if (head != nullptr)

{

int choice = 1;

/\*cout << "Введите 1 - вывод списка с начала\nВведите 2 - вывод списка с конца\n\n";

cin >> choice;

system("cls");\*/

switch (choice)

{

case 1:

{

/\*cout << "Список: ";\*/

Node\* current = head;

for (size\_t i = 0; i < size; i++)

{

cout << current->data << " ";

current = current->pnext;

}

cout << endl;

break;

}

case 2:

{

cout << "Список: ";

Node\* current = tail;

for (size\_t i = size; i > 0; i--)

{

cout << current->data << " ";

current = current->pprev;

}

cout << endl;

break;

}

default:

system("cls"); cout << "Ошибка ввода!\n";

break;

}

}

else { system("cls"); cout << "Список пуст!\n"; }

}

void List::add\_after(int number, int data)

{

if (head != nullptr)

{

if (number >= 0 && number <= size)

{

if (number == size)

push\_back(data);

else if (number == 0)

push\_front(data);

else

{

if (number <= size / 2)

{

Node\* current = head;

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

current = current->pnext;

Node\* temp = current->pnext;

current->pnext = new Node(data);

current->pprev = current;

current = current->pnext;

current->pnext = temp;

++size;

}

else

{

Node\* current = tail;

for (int i = size; i > number; i--)

current = current->pprev;

Node\* temp = current->pnext;

Node\* temp2 = current;

current->pnext = new Node(data);

temp->pprev = current->pnext;

current = current->pnext;

current->pprev = temp2;

current->pnext = temp;

++size;

}

}

}

else { system("cls"); cout << "Ошибка ввода!\n\n"; }

}

else { system("cls"); cout << "Список пуст!\n\n"; }

}

void List::pop\_element(int number)

{

if (head != nullptr)

{

if (number >= 1 && number <= size)

{

if (number == size)

pop\_back();

else if (number == 1)

pop\_front();

else

{

if (number <= size / 2)

{

Node\* current = head;

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

current = current->pnext;

Node\* temp = current->pprev;

Node\* temp2 = current->pnext;

delete current;

temp->pnext = temp2;

temp2->pprev = temp;

}

else

{

Node\* current = tail;

for (int i = size; i > number; i--)

current = current->pprev;

Node\* temp = current->pprev;

Node\* temp2 = current->pnext;

delete current;

temp->pnext = temp2;

temp2->pprev = temp;

}

--size;

}

}

else { system("cls"); cout << "Ошибка ввода!\n\n"; }

}

else { system("cls"); cout << "Список пуст!\n\n"; }

}

int& List::operator[](const int index) const

{

if (index >= 0 && index <= size)

{

if (index <= size / 2)

{

Node\* current = head;

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

current = current->pnext;

return current->data;

}

else

{

Node\* current = tail;

for (int i = index; i > index; --i)

current = current->pprev;

return current->data;

}

}

}

int& List::operator()()

{

return size;

}

List& List::operator=(const List& object)

{

if (this != &object)

if (this->head != nullptr)

{

this->~List();

for (int i = 0; i < object.size; i++)

this->push\_back(object[i]);

}

else

for (int i = 0; i < object.size; i++)

this->push\_back(object[i]);

return \*this;

}

List List::operator\*(List& object)

{

if (object.size > size)

{

List temp;

Node\* current = (\*this).head;

for (int i = 0; i < object.size; i++)

{

if (current != nullptr)

temp.push\_back((\*this)[i] \* object[i]);

else

temp.push\_back(0);

if (current != 0)

current = current->pnext;

}

return temp;

}

else

{

List temp;

Node\* current = object.head;

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

{

if (current != nullptr)

temp.push\_back((\*this)[i] \* object[i]);

else

temp.push\_back(0);

if (current != 0)

current = current->pnext;

}

return temp;

}

}

Class Iterator

#include "List.h"

List::Iterator::Iterator()

{

it = nullptr;

}

List::Iterator::Iterator(const Iterator& it)

{

this->it = it.it;

}

bool List::Iterator::operator==(const Iterator& it)

{

return this->it = it.it;

}

bool List::Iterator::operator!=(const Iterator& it)

{

return this->it != it.it;

}

void List::Iterator::operator++()

{

it = it->pnext;

}

void List::Iterator::operator--()

{

it = it->pprev;

}

int List::Iterator::operator\*()

{

return this->it->data;

}

Class Pair

#include "Pair.h"

Pair::Pair()

{

first = 0;

second = 0;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair::Pair(const Pair& ob)

{

this->first = ob.first;

this->second = ob.second;

}

Pair::~Pair()

{

}

Pair Pair::operator-(Pair& pair) const

{

Pair temp = \*this;

temp.first -= pair.first;

temp.second -= pair.second;

return temp;

}

Pair Pair::operator+(Pair& pair) const

{

Pair temp = \*this;

temp.first += pair.first;

temp.second += pair.second;

return temp;

}

Pair Pair::operator+(int data) const

{

Pair temp = \*this;

temp.first += data;

return temp;

}

Pair Pair::operator+(double data) const

{

Pair temp = \*this;

temp.second += data;

return temp;

}

Pair& Pair::operator++()

{

++first;

++second;

return \*this;

}

Pair Pair::operator++(int)

{

Pair temp = \*this;

++first;

++second;

return temp;

}

Pair Pair::operator\*(Pair& pair)

{

Pair temp;

temp.first = this->first \* pair.first;

temp.second = this->second \* pair.second;

return temp;

}

Pair& Pair::operator=(const Pair& pair)

{

if (this == &pair)

return \*this;

first = pair.first;

second = pair.second;

return \*this;

}

ostream& operator<<(ostream& out, const Pair& ob)

{

out << "(" << ob.first << ":" << ob.second << ")";

return out;

}

istream& operator>>(istream& in, Pair& ob)

{

cout << "first(int): "; in >> ob.first;

cout << "second(double): "; in >> ob.second;

return in;

}

**Результаты работы программы**

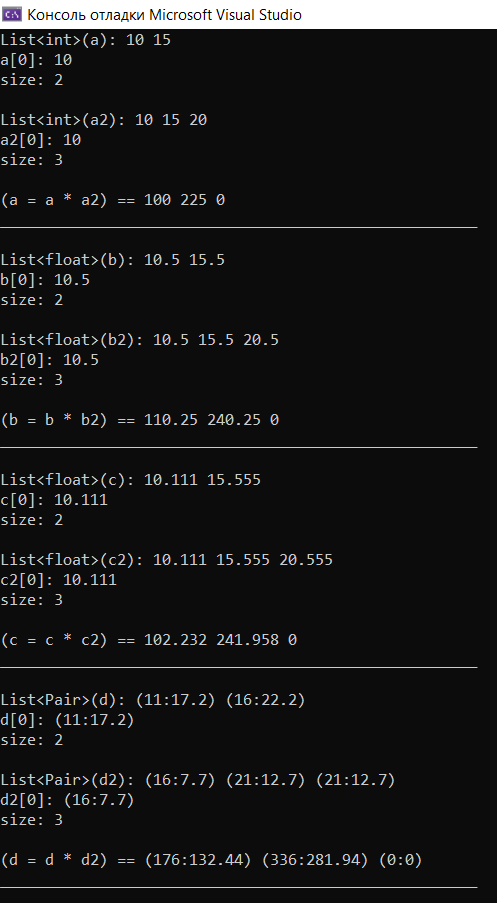


Рисунок 2 – результат работы программы