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**

Тема: «АТД. Контейнеры»

Выполнил работу

Студент группы РИС-22-2Б

Малых В.О.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь 2023

**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.

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 класса.

Класс- контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции:

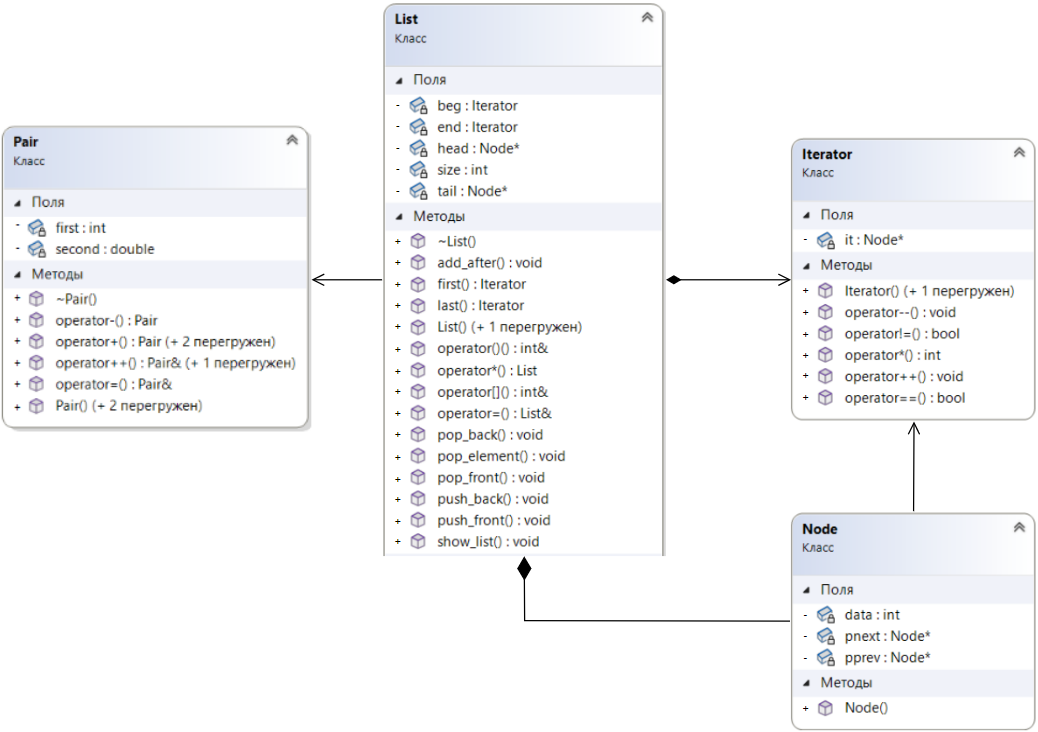
[ - доступа по индексу;

int() - определение размера списка;

\* вектор - умножение элементов списков а[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.

**UML - диаграмма**



**Код программы.**

Sourse.cpp

#include "List.h"

#include "Pair.h"

int main()

{

system("chcp 1251 >> null");

List<int> a, b, c;

for (int i = 0; i < 5; i++)

{

a.push\_back(i + 2);

}

for (int i = 0; i < 4; i++)

{

b.push\_back(i-1);

}

cout << "Лист а:\n";

a.show\_list();

cout << "\nЛист b:\n";

b.show\_list();

cout << "\nЛист с = a\*b :\n";

c = a \* b;

c.show\_list();

cout << "\nРазмер листа c: " << c() << endl << endl;

cout << "\nприсвоить c[5] = 150" << endl; c[5] = 150;

cout << "\nЛист с:\n";

c.show\_list();

List<Pair> p;

Pair x;

for (int i = 0; i < 3; i++)

{

cout << "\nВведите пару:\n";

cin >> x;

p.push\_back(x);

}

cout << "\nСписок p: ";

p.show\_list();

return 0;

}

List.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

template <typename T>

class List

{

private:

class Node

{

friend class List;

private:

T data;

Node\* pnext = nullptr;

Node\* pprev = nullptr;

public:

Node(T 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(T data);

void pop\_front();

void push\_back(T data);

void pop\_back();

void show\_list();

void add\_after(int number, T 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;

}

};

template <typename T>

List<T>::List()

{

size = 0;

head = nullptr;

tail = nullptr;

}

template <typename T>

List<T>::List(List& object)

{

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

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

}

template <typename T>

List<T>::~List()

{

if (head != nullptr)

{

Node\* current = head;

while (current != nullptr)

{

Node\* next = current->pnext;

delete current;

current = next;

}

head = nullptr;

tail = nullptr;

size = 0;

}

}

template <typename T>

List<T>::Node::Node(T data)

{

this->data = data;

}

template <typename T>

void List<T>::push\_front(T 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;

}

template <typename T>

void List<T>::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";

}

}

template <typename T>

void List<T>::push\_back(T 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;

}

template <typename T>

void List<T>::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"; }

}

template <typename T>

void List<T>::show\_list()

{

if (head != nullptr)

{

int choice = 1;

switch (choice)

{

case 1:

{

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"; }

}

template <typename T>

void List<T>::add\_after(int number, T 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"; }

}

template <typename T>

void List<T>::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"; }

}

template <typename T>

int& List<T>::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 > size/2+1; --i) /////

current = current->pprev;

return current->data;

}

}

}

template <typename T>

int& List<T>::operator()()

{

return size;

}

template <typename T>

List<T>& List<T>::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;

}

template <typename T>

List<T> List<T>::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;

}

}

template <typename T>

List<T>::Iterator::Iterator()

{

it = nullptr;

}

template <typename T>

List<T>::Iterator::Iterator(const Iterator& it)

{

this->it = it.it;

}

template <typename T>

bool List<T>::Iterator::operator==(const Iterator& it)

{

return this->it = it.it;

}

template <typename T>

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

{

return this->it != it.it;

}

template <typename T>

void List<T>::Iterator::operator++()

{

it = it->pnext;

}

template <typename T>

void List<T>::Iterator::operator--()

{

it = it->pprev;

}

template <typename T>

int List<T>::Iterator::operator\*()

{

return this->it->data;

}

Pair.h

#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 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=(const Pair& pair);

~Pair();

};

Pair.cpp

#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::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=(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;

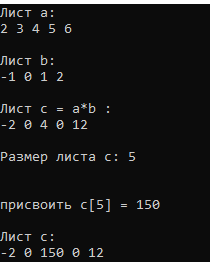
}

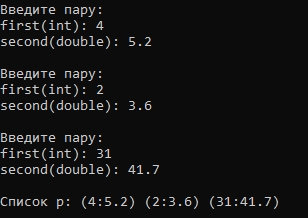
Pair::~Pair()

{

}

Вывод





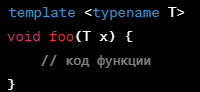
Контрольные вопросы.

1. В чем смысл использования шаблонов?

Использование шаблонов позволяет создавать обобщенный код, который может быть использован с различными типами данных или объектами без необходимости написания отдельного кода для каждого типа или объекта. Это упрощает разработку и поддержку кода.

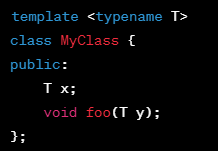
1. Каковы синтаксис/семантика шаблонов функций?

Синтаксис шаблонов функций заключается в использовании ключевого слова "template" перед определением функции и в указании параметров шаблона в угловых скобках.



1. Каковы синтаксис/семантика шаблонов классов?

Синтаксис шаблонов классов похож на синтаксис шаблонов функций, но параметры шаблона указываются после имени класса.



Здесь "T" - это параметр шаблона класса, который может быть заменен на любой тип данных при создании объектов класса.

1. Что такое параметры шаблона функции?

Параметры шаблона функции - это типы данных или константы, которые могут быть переданы в функцию в качестве аргументов. Они используются для создания обобщенного кода, который может работать с различными типами данных.

1. Перечислите основные свойства параметров шаблона функции.

Основные свойства параметров шаблона функции:

* Они могут быть заменены на любой тип данных при вызове функции.
* Они могут быть константами или типами данных.
* Они могут иметь значения по умолчанию.

1. Как записывать параметр шаблона?

Параметр шаблона записывается в угловых скобках после ключевого слова "template". Например: **template <typename T>**

1. Можно ли перегружать параметризованные функции?

Параметризованные функции могут быть перегружены, как и обычные функции. Они должны отличаться друг от друга типами параметров шаблона или аргументами функции.

1. Перечислите основные свойства параметризованных классов.

Основные свойства параметризованных классов:

* Они позволяют создавать обобщенные классы, которые могут работать с различными типами данных.
* Они могут содержать параметры шаблона для определения типов данных, используемых внутри класса.
* Они могут быть унаследованы от других параметризованных классов или обычных классов.

1. Все ли компонентные функции параметризованного класса являются параметризованными?

Не все компонентные функции параметризованного класса должны быть параметризованными. Некоторые функции могут использовать обычные типы данных, не зависящие от параметров шаблона.

1. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?

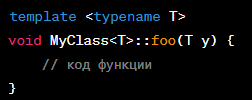
Дружественные функции, описанные в параметризованном классе, могут быть параметризованными или не параметризованными, в зависимости от того, используют ли они параметры шаблона класса или обычные типы данных. Если дружественная функция использует параметры шаблона класса, то она должна быть также параметризованной.

1. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Шаблоны классов могут содержать виртуальные компонентные функции.

1. Как определяются компонентные функции параметризованных классов вне определения шаблона класса?

Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса, так же как и для обычных классов. Однако при определении функции вне шаблона класса необходимо указать параметры шаблона перед именем класса.



1. Что такое инстанцирование шаблона?

Инстанцирование шаблона - это процесс создания экземпляра класса или функции на основе определения шаблона с конкретными параметрами шаблона. Например, если у нас есть шаблон класса **MyClass**, мы можем создать экземпляр этого класса с конкретным типом данных.

1. На каком этапе происходит генерирование определения класса по шаблону?

Генерирование определения класса по шаблону происходит на этапе компиляции. Когда компилятор встречает использование шаблона класса или функции в коде, он генерирует соответствующее определение на основе шаблона с конкретными параметрами шаблона. Это происходит перед созданием объектов или вызовом функций с использованием шаблона.