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

ПНИПУ

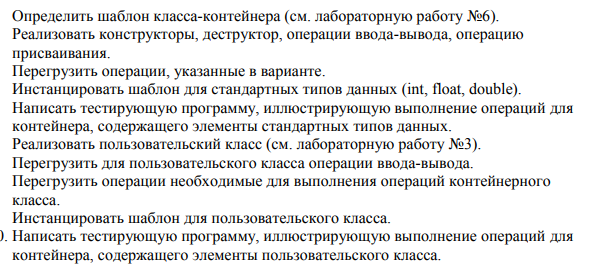
**Лабораторная работа  
“Классы”. №7**

Выполнил:   
студент группы РИС-23-1б   
Сингур Иван Сергеевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

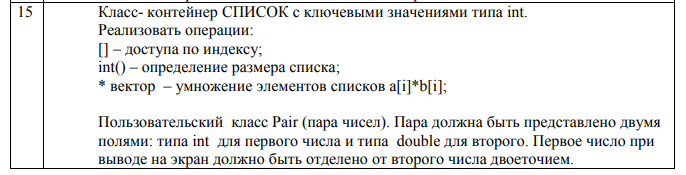
Пермь, 2024 г.

**«Классы и объекты. Шаблоны классов»**



**Анализ задачи:**

***Вариант 15:***



**Код на языке C++:**

#include <iostream>

#include <string>

using namespace std;

template <typename T>

struct Node {

T data;

Node<T>\* next = nullptr;

Node<T>\* prev = nullptr;

};

template <typename T>

class List {

private:

int size;

Node<T>\* head;

Node<T>\* tail;

public:

void pushFront(T data) {

Node<T>\* newNode = new Node<T>;

newNode->data = data;

if (head == nullptr) {

this->head = newNode;

this->tail = newNode;

}

else {

head->prev = newNode;

newNode->next = head;

head = newNode;

}

this->size++;

}

void pushBack(T data) {

Node<T>\* newNode = new Node<T>;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

this->head = newNode;

this->tail = newNode;

}

else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

this->size++;

}

List<T>(int size) {

this->size = size;

if (size > 0) {

Node<T>\* node = new Node<T>;

this->head = node;

this->tail = node;

for (int i = 1; i < size; i++) {

Node<T>\* newNode = new Node<T>;

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

else {

this->head = nullptr;

this->tail = nullptr;

}

}

List<T>(int size, T data) {

this->size = size;

if (size > 0) {

Node<T>\* node = new Node<T>;

node->data = data;

this->head = node;

this->tail = node;

for (int i = 1; i < size; i++) {

Node<T>\* newNode = new Node<T>;

newNode->data = data;

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

else {

this->head = nullptr;

this->tail = nullptr;

}

}

List<T>(const List<T>& list) {

this->head = nullptr;

this->tail = nullptr;

this->size = 0;

Node<T>\* current\_node = list.head;

while (current\_node != nullptr) {

pushBack(current\_node->data);

current\_node = current\_node->next;

}

}

~List<T>() {

Node<T>\* node = head;

while (node != nullptr) {

Node<T>\* newNode = node->next;

delete node;

node = newNode;

}

head = nullptr;

}

T popBack() {

T temp = 0;

if (this->tail != nullptr) {

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

tail = current\_node->prev;

temp = current\_node->data;

tail->next = nullptr;

this->size = size - 1;

delete current\_node;

}

return temp;

}

T popFront() {

T temp=0;

if (this->head != nullptr) {

Node<T>\* current\_node = this->head;

head = current\_node->next;

temp = current\_node->data;

if (head) {

head->prev = nullptr;

}

delete current\_node;

this->size = size - 1;

}

return temp;

}

T front() {

return this->head->data;

}

T back() {

return this->tail->data;

}

bool is\_empty() {

return this->size == 0;

}

List<T>& operator =(const List<T>& list) {

cout << "Вызвана операция присваивания: " << endl;

if (this == &list) {

return \*this;

}

while (head != nullptr) {

Node<T>\* temp = head;

head = head->next;

delete temp;

}

this->size = 0;

Node<T>\* current = list.head;

while (current != nullptr) {

pushBack(current->data);

current = current->next;

}

return \*this;

}

T& operator [] (int index) {

cout << "Вызвана операция поиска по индексу: " << endl;

if (index < this->size && index >= 0) {

Node<T>\* current = head;

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

current = current->next;

}

return current->data;

}

else {

cout << "Индекс не найден " << endl;

}

}

int operator () () {

return this->size;

}

List<T> operator \* (List<T>& list) {

int temp;

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

temp = list.size;

}

else {

temp = this->size;

}

List<T> templ(temp, 0);

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

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

}

return templ;

}

friend istream& operator >>(istream& is,List<T>& list) {

cout << "Вызван оператор перегрузки >>" << endl << endl;

Node<T>\* current\_node = list.head;

while (current\_node != nullptr) {

is >> current\_node->data;

current\_node = current\_node->next;

}

cout << "Ввод закончен" << endl << endl;

return is;

}

friend ostream& operator <<(ostream& os,const List<T>& list) {

cout << "Вызван оператор перегрузки << " << endl << endl;

Node<T>\* current\_node = list.head;

while (current\_node != nullptr) {

os << current\_node->data << " ";

current\_node = current\_node->next;

}

cout << endl << endl << "Вывод закончен" << endl << endl;

return os;

}

};

template <typename T1, typename T2>

class Pair {

private:

T1 x;

T2 y;

public:

Pair <T1,T2>() {

this->x = 0;

this->y = 0;

cout << "Вызван конструктор без параметров: " << this << endl;

cout << "X равно: " << this->x << '\t' << "Y равно: " << this->y << endl << endl;

}

Pair<T1, T2>(T1 i, T2 j) {

this->x = i;

this->y = j;

cout << "Вызван конструктор с параметрами: " << this << endl;

cout << "X равно: " << this->x << '\t' << "Y равно: " << this->y << endl << endl;

}

Pair<T1, T2>(const Pair& d) {

this->x = d.x;

this->y = d.y;

cout << "Вызван конструктор копирования: " << this << endl;

cout << "X равно: " << this->x << '\t' << "Y равно: " << this->y << endl << endl;

}

T1 diff() {

return this->x - this->y;

}

void dobavlenieConst(T1 j) {

cout << "К " << this->x << " добавляем const " << j << ": " << endl;

this->x = x + j;

}

void dobavlenieConst(T2 i) {

cout << "К " << this->y << " добавляем const " << i << ": " << endl;

this->y = y + i;

}

void GetInfo() {

cout << this->x << " : " << this->y << endl;

}

void operator=(const Pair<T1, T2>& a) {

this->x = a.x;

this->y = a.y;

}

void setX(T1 b) {

this->x = b;

}

void setY(T2 a) {

this->y = a;

}

T1 returnX() {

return this->x;

}

T2 returnY() {

return this->y;

}

void operator -(const Pair<T1, T2>& a) {

this->x = x - a.x;

this->y = y - a.y;

}

void operator +(T1 j) {

cout << "К " << this->x << " добавляем const " << j << ": " << endl;

this->x = x + j;

}

void operator +(T2 k) {

cout << "К " << this->y << " добавляем const " << k << ": " << endl;

this->y = y + k;

}

friend ostream& operator << (ostream& os, Pair<T1, T2>& p) {

cout << "Вызвана перегрузка оператора вывода: " << endl << endl;

os << p.x << " : " << p.y << endl << endl;

cout << "Перегрузка закончилась" << endl << endl;

return os;

}

friend istream& operator >> (istream& is, Pair<T1, T2>& p) {

cout << "Вызвана перегрузка оператора ввода: " << endl << endl;

is >> p.x;

is >> p.y;

cout << "Перегрузка закончилась" << endl << endl;

return is;

}

~Pair<T1, T2>() {

cout << "Вызван деструктор: " << this << endl;

}

};

int main() {

system("chcp 1251 > NULL");

List<int> l1(4);

l1.pushBack(3);

l1.pushBack(4);

l1.pushBack(5);

l1.pushBack(6);

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

l1.popFront();

}

cout << "Первый список: " << endl << endl;

cout << l1;

List<string> l2(4);

l2.pushBack("Димка");

l2.pushBack("Чунарев");

l2.pushBack("Привет");

l2.pushBack("Бро");

cout << "Второй список: " << endl << endl;

cout << l2;

List<float> l3(4);

l3.pushBack(3.2);

l3.pushBack(4.3);

l3.pushBack(5.4);

l3.pushBack(6.5);

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

l3.popFront();

}

cout << "Третий список: " << endl << endl;

cout << l3;

cout << endl << endl;

cout << "Работа с классом Pair: " << endl << endl;

Pair<double, int> p1;

cin >> p1;

cout << "Добавление const к x: " << endl;

p1.dobavlenieConst(5.243);

cout << "Добавление const к y: " << endl;

p1.dobavlenieConst(7);

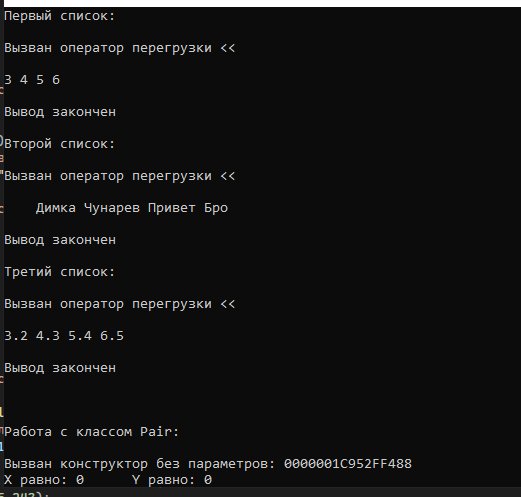
cout << p1;

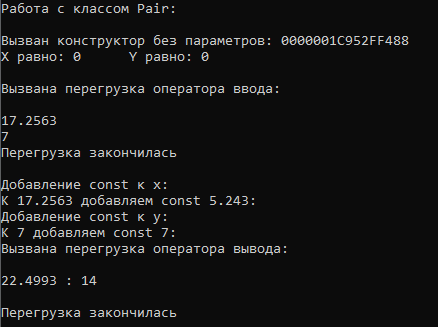
cout << endl << endl;

return 0;

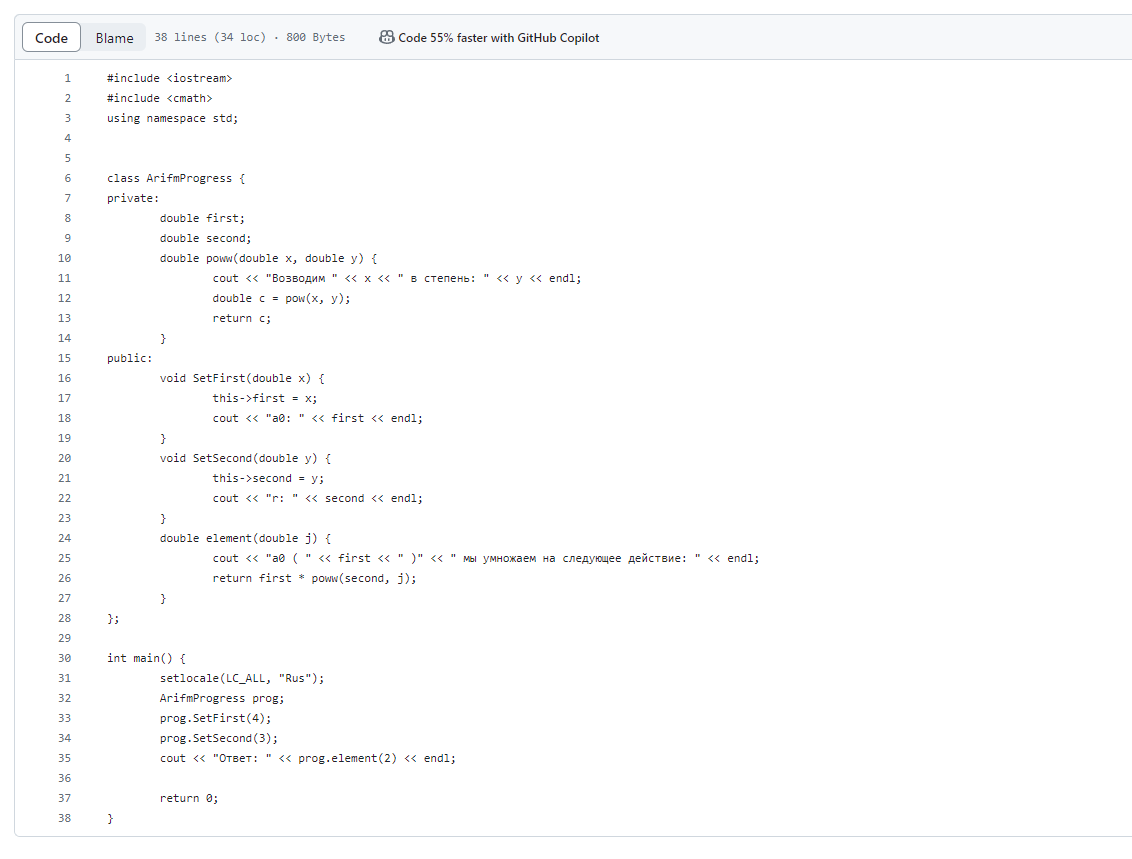
}

**Работа программы:**

****



**Cкрины из гита:**



**Ссылка на гит:**

<https://github.com/elleoon/pnipu1>

**Ответ на вопросы:**

1. Использование шаблонов в C++ позволяет программистам создавать обобщенные конструкции, которые могут работать с различными типами данных без необходимости создания отдельных версий функций или классов для каждого типа. Шаблоны обеспечивают гибкость и повторное использование кода.

2. Синтаксис и семантика шаблонов функций в C++:

- Синтаксис: `template <typename T> void functionName(T parameter)`

- Шаблонная функция начинается с ключевого слова `template`, за которым идет объявление шаблонных параметров заключенных в угловые скобки `< >`.

3. Синтаксис и семантика шаблонов классов в C++:

- Синтаксис: `template <class T> class ClassName { /\* ... \*/ };`

- Шаблонный класс начинается с ключевого слова `template`, за которым идет объявление шаблонных параметров.

4. Параметры шаблона функции - это типы данных или значения, которые передаются в шаблонную функцию для использования внутри тела функции.

5. Основные свойства параметров шаблона функции включают:

- Возможность использования различных типов данных или значений при вызове функции.

- Универсальность и повторное использование кода.

6. Параметры шаблона записываются в угловых скобках после ключевого слова `template`, например: `template <typename T>`.

7. Да, можно перегружать параметризованные функции путем изменения параметров шаблона, типов данных или количества параметров.

8. Основные свойства параметризованных классов включают возможность создания классов, которые могут работать с различными типами данных без необходимости создания отдельных версий для каждого типа.

9. Нет, не все компонентные функции параметризованного класса обязательно должны быть параметризованными.

10. Дружественные функции, описанные в параметризованном классе, также могут быть параметризованными.

11. Да, шаблоны классов могут содержать виртуальные компонентные функции.

12. Компонентные функции параметризованных классов могут быть определены вне шаблона класса с помощью специализации шаблона.

13. Инстанцирование шаблона - это процесс создания конкретной реализации класса или функции из шаблона с определенными типами данных.

14. Генерирование определения класса по шаблону происходит во время компиляции, когда требуется создание инстанции шаблона с определенными типами.

**Вывод:** Задача была выполнена. Всё получилось.

