Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

**Тема:** Лабораторная работа № 7 по ООП

Семестр: 2

Выполнил студент ИВТ-22-2б:

Солодов Александр Андреевич

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

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 класса.

*Вариант 12*

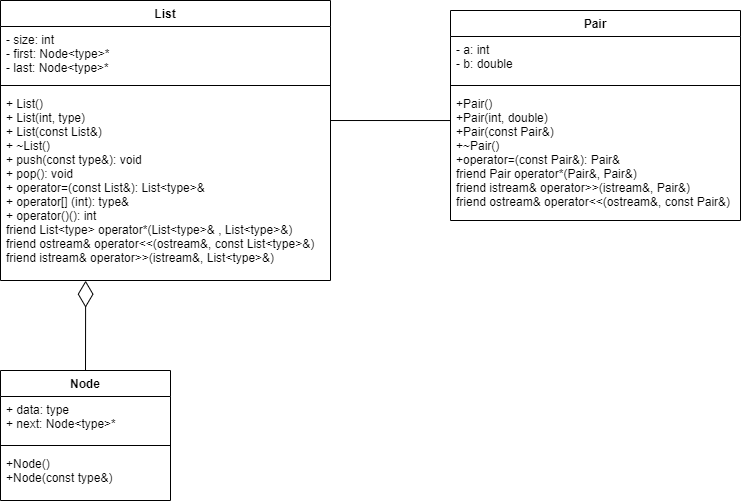
[] – доступа по индексу;

() – определение размера списка;

+ число – добавляет константу ко всем элементам вектора;

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**

****

**Программное решение**

List.h

#pragma once

#include <iostream>

using namespace std;

template <typename T>

class Node

{

public:

T data;

Node\* pNext;

Node(T data = T(), Node\* pNext = nullptr) {

this->data = data;

this->pNext = pNext;

}

};

template <typename T>

class List

{

private:

int size;

Node<T>\* first;

Node<T>\* last;

public:

List();

List(int, T);

List(const List<T>&);

~List();

void push\_back(T&);

void pop\_front();

//перегрузки операторов

List<T>& operator = (const List<T>&);

T& operator [] (int);

int operator () ();

void operator + (const int);

//перегрузки операций ввода и вывода

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

int count;

cout << "Введите размер списка: ";

in >> count;

cout << "Введите элементы списка через Enter: " << endl;

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

T data;

in >> data;

list.push\_back(data);

}

return in;

}

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

if (list.first == nullptr) {

cout << "Список пуст!" << endl;

return out;

}

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

while (current != nullptr) {

cout << '(' << current->data << ')' << " ";

current = current->pNext;

}

cout << endl;

return out;

}

};

List.cpp

#include "List.h"

#include <iostream>

using namespace std;

//конструктор по умолчанию

template <typename T>

List<T>::List() {

size = 0;

first = nullptr;

last = nullptr;

}

//конструктор с параметрами

template <typename T>

List<T>::List(int s, T d) {

size = s;

first = nullptr;

last = nullptr;

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

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

addNode->data = d;

if (last == nullptr) {

first = addNode;

last = addNode;

}

else {

last->pNext = addNode;

last = addNode;

}

}

}

//конструктор копирования

template <typename T>

List<T>::List(const List<T>& L) {

size = 0;

first = nullptr;

last = nullptr;

Node<T>\* current = L.first;

while (current != nullptr) {

Node<T>\* addNode = new Node<T>();

addNode->data = current->data;

addNode->pNext = nullptr;

if (last == nullptr) {

first = addNode;

last = addNode;

}

else {

last->next = addNode;

last = addNode;

}

current = current->next;

size++;

}

}

//деструктор

template <typename T>

List<T>::~List()

{

while (size > 0) {

pop\_front();

}

}

//функция для добавления элемента в конец списка

template <typename T>

void List<T>::push\_back(T& d) {

if (first == nullptr) {

first = new Node<T>(d);

}

else {

Node<T>\* current = this->first;

while (current->pNext != nullptr) {

current = current->pNext;

}

current->pNext = new Node<T>(d);

}

size++;

}

//функция для удаления элемента из начала списка

template <typename T>

void List<T>::pop\_front() {

if (first == nullptr) {

cout << "Список пуст!" << endl;

return;

}

Node<T>\* temp = first;

first = first->pNext;

delete temp;

size--;

}

//перегрузка оператора присваивания

template <typename T>

List<T>& List<T>::operator = (const List<T>& other)

{

if (this == &other) {

return \*this;

}

while (first != nullptr) {

Node<T>\* temp = first;

first = first->pNext;

delete temp;

}

Node<T>\* currentOther = other.first;

while (currentOther != nullptr) {

push\_back(currentOther->data);

currentOther = currentOther->pNext;

}

return \*this;

}

//перегрузка квадратных скобок

template <typename T>

T& List<T>::operator [] (int index) {

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

int count = 0;

Node<T>\* current = this->first;

while (current != nullptr) {

if (count == index) {

return current->data;

}

current = current->pNext;

count++;

}

}

else {

cout << "Некорректный индекс!" << endl;

}

}

//перегрузка круглых скобок - получение размера списка

template <typename T>

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

return size;

}

//перегрузка оператора сложения

template <typename T>

void List<T>::operator + (const int add) {

Node<T>\* current = this->first;

while (current != nullptr) {

current->data = current->data + add;

current = current->pNext;

}

}

Pair.h

#pragma once

#include <iostream>

using namespace std;

class Pair {

private:

int left;

double right;

public:

Pair() {

left = 0;

right = 0.0;

}

Pair(int l, double r) {

left = l;

right = r;

}

Pair(const Pair& p) {

left = p.left;

right = p.right;

}

~Pair() {}

//геттеры

int get\_left() {

return left;

}

void set\_left(int l) {

left = l;

}

//сеттеры

double get\_right() {

return right;

}

void set\_right(double r) {

right = r;

}

//перегрузки операций ввода-вывода

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

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

//перегрузки операторов сравнения

void operator < (const Pair& p);

void operator > (const Pair& p);

//перегрузка префиксного инкремента

void operator ++ () { ++left; }

//перегрузка постфиксного инкремента

void operator ++ (int) { right += 1.0; }

void operator + (const int);

};

Pair.cpp

#include "Pair.h"

//перегрузка операции ввода

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

cout << "Введите левое число: ";

in >> p.left;

cout << "Введите правое число: ";

in >> p.right;

return in;

}

//перегрузка операции вывода

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

return (out << p.left << ":" << p.right);

}

//перегрузка операций сравнения

void Pair::operator < (const Pair& p) {

if ((this->left < p.left) || (this->left == p.left && this->right < p.right)) {

cout << "Первая пара чисел меньше, чем вторая пара" << endl;

}

else {

cout << "Первая пара чисел НЕ меньше, чем вторая пара" << endl;

}

}

void Pair::operator > (const Pair& p) {

if ((this->left > p.left) || (this->left == p.left && this->right > p.right)) {

cout << "Первая пара чисел больше, чем вторая пара" << endl;

}

else {

cout << "Первая пара чисел НЕ больше, чем вторая пара" << endl;

}

}

void Pair::operator + (const int add) {

left += add;

right += add;

}

main.cpp

/\* Вариант 12.

Класс-контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции:

[] – доступа по индексу;

() – определение размера списка;

+ число – добавляет константу ко всем элементам списка;

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.

\*/

#include "List.h"

#include "List.cpp"

#include "Pair.h"

int main() {

setlocale(LC\_ALL, "RU");

//создание списка строк через коструктор с параметрами

List<string> list1(4, "list");

cout << "Первый список: " << list1 << endl;

//создание второго списка с помощью ввода

List<int> list2;

cin >> list2;

cout << "Второй список: " << list2 << endl;

//проверка перегрузок [] и ()

int ind1;

cout << "Введите индекс элемента для отображения: ";

cin >> ind1;

cout << "Элемент с индексом " << ind1 << " это " << list2[ind1] << endl;

cout << "Размер второго списка: " << list2() << endl;

//доабвление константы ко всем элементам второго списка

list2 + 5;

cout << "Второй список после добавления константы 5: " << list2 << endl;

//создание пар чисел с помощью ввода

Pair pair1, pair2;

cout << "Ввод первой пары чисел: " << endl;

cin >> pair1;

cout << "Ввод второй пары чисел: " << endl;

cin >> pair2;

cout << "Первая пара: " << pair1 << endl << "Вторая пара: " << pair2 << endl;

//перегрузка операции сравнения для пар

pair1 > pair2;

cout << endl;

//создание списков с элементами в виде пар чисел

List<Pair> list3, list4;

cin >> list3;

cout << "Третий список: " << list3 << endl;

//копирование третьего списка в четвёртый

list4 = list3;

cout << "Четвёртый список, скопированный из третьего: " << list3 << endl;

cout << "Введите индекс элемента для отображения: ";

//проверка перегрузок [] и ()

int ind2;

cin >> ind2;

cout << "Элемент с индексом " << ind2 << " это " << "(" << list4[ind2] << ")" << endl;

cout << "Размер четвёртого списка: " << list4() << endl;

//добавление константы ко всем элементам четвертого списка

for (int i = 0; i < list4(); i++) {

list4[i] + 5;

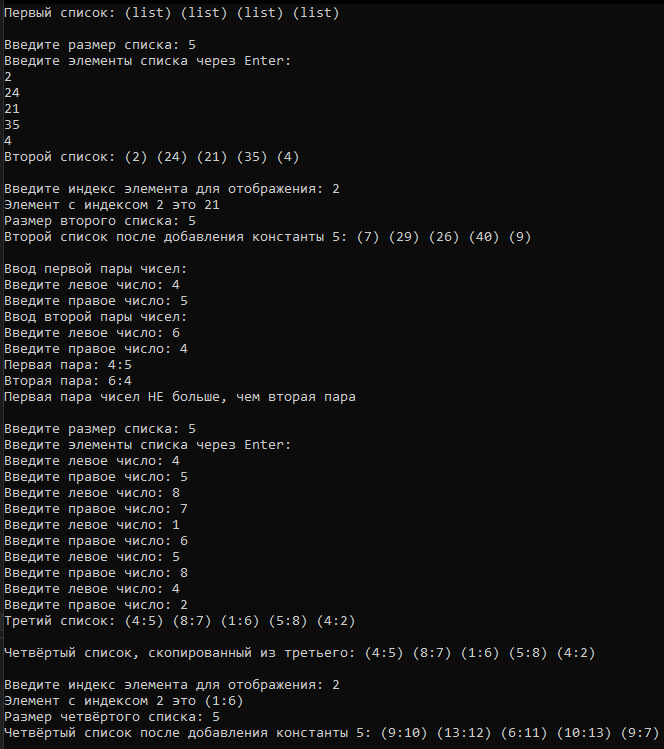
}

cout << "Четвёртый список после добавления константы 5: " << list4 << endl;

return 0;

}

**Результат работы программы**



**Контрольные вопросы**

*1. В чем смысл использования шаблонов?*

С помощью шаблона функций можно отделить алгоритм от конкретных типов данных, передавая тип в качестве параметра. Шаблоны классов предоставляют аналогичную возможность, позволяя создавать параметризированные классы.

*2. Каковы синтаксис/семантика шаблонов функций?*

Формат шаблона :

template <параметры\_шаблона>

заголовок\_функции

{ тело функции }

*Пример:*

template<class type>//type – имя параметризируемого типа

type abs(type x)

{

if (x < 0)return -x;

else return x;

}

*3. Каковы синтаксис/семантика шаблонов классов?*

template <параметры шаблона>

class имя\_класса

{…};

*Пример:*

template<classT>

class Point

{

T x, y;//координаты точки

public:

Point(T X = 0, T Y = 0) :x(X), y(Y) {}

void Show();

};

template<classT>

void Point::Show()

{

cout << ”(“ << x << ”, ” << y << ”)”;

}

};

*4. Что такое параметры шаблона функции?*

Можно считать, что параметры шаблона являются его формальными параметрами, а типы тех параметров, которые используются в конкретных обращениях к функции, служат фактическими параметрами шаблона. Именно по ним выполняется параметрическая настройка и с учетом этих типов генерируется конкретный текст определения функции.  
Если в программе вызов функции осуществляется как abs(-1.5), то компилятор формирует определение функции double abs(double x)

*Пример:*

template <typename T>

T abs(T x) {

if (x > 0) return x;

else return -x;

}

*5. Перечислите основные свойства параметров шаблона функции.*

- Имена параметров шаблона должны быть уникальными во всем определении шаблона.  
- Список параметров шаблона не может быть пустым, для того, чтобы компилятор мог инстанцировать шаблон.  
- В списке параметров шаблона может быть несколько параметров, и каждому из них должно предшествовать ключевое слово class.  
template<class type1, class type2>  
Соответственно, неверен заголовок:  
template<class type1, type2, type3>  
- Недопустимо использовать в заголовке шаблона параметры с одинаковыми именами, то есть ошибочен такой заголовок:  
template<class t, class t, class t>  
- Имя параметра шаблона имеет все права имени типа в определенной шаблоном функции.  
- Все параметры шаблона функций должны быть обязательно использованы в спецификациях параметров определения функции. Таким образом, будет ошибочным такой шаблон:

template<class A, class B, class C>

B func(A n, C m) { B value; };

- Определенная с помощью шаблона функция может иметь любое количество непараметризованных формальных параметров. Может быть непараметризованно и возвращаемое функцией значение. Например, в следующей программе шаблон определяет семейство функций, каждая из которых подсчитывает количество нулевых элементов одномерного массива параметризованного типа:

template<class D> long count0(int, D\*); //Прототип шаблона

int main(void) {

int A[] = { 1, 0, 6, 0, 4, 10 };

int n = sizeof(A) / sizeof A[0];

cout « "\ncount0(n,A) = " « count0(n, A);

float X[] = { 10.0, 0.0, 3.3, 0.0, 2.1 };

n = sizeof(X) / sizeof X[];

cout « "\ncount0(n,X) = " « count0(n, X);

return 0;

}  
- В списке параметров прототипа шаблона имена параметров не обязаны совпадать с именами тех же параметров в определении шаблона.  
- При конкретизации шаблонного определения функции необходимо, чтобы при вызове функции типы фактических параметров, соответствующие одинаково параметризованным формальным параметрам, были одинаковыми. Для определенного ниже шаблона функций с данным прототипом недопустимо использовать такое обращение к функции:

template<class E> void swap(E, E);

int n = 4;

double d = 4.3;

swap(n, d); // Ошибка в типах параметров

swap(double(n), d); // Правильные типы параметров

Для правильного обращения к такой функции требуется явное приведение типа одного из параметров.  
- При использовании шаблонов функций возможна перегрузка как шаблонов, так и функций. Могут быть шаблоны с одинаковыми именами, но разными параметрами. Или с помощью шаблона может создаваться функция с таким же именем, что и явно определенная функция. В обоих случаях "распознавание" конкретного вызова выполняется по сигнатуре, т.е. по типам, порядку и количеству фактических параметров.

*6. Как записывать параметр шаблона?*

- Каждому из разных типов параметров должно предшествовать ключевое слово class. Пример: template<class type1, class type2>  
- Все параметры шаблона функций должны быть обязательно использованы в спецификациях параметров определения функции.  
- Недопустимо использовать в заголовке шаблона параметры с одинаковыми именами, то есть ошибочен такой заголовок:  
template<class t, class t, class t>

*7. Можно ли перегружать параметризованные функции?*

Шаблон функции может перегружать функции, отличные от шаблона, с тем же именем. В этом сценарии компилятор сначала пытается разрешить вызов функции с помощью вычета аргументов шаблона для создания экземпляра шаблона функции с уникальной специализацией.

*Пример:*

template <typename T>

void print(T value) { cout « “Шаблон” « value; }

template <>

void print(int value) { cout « “Специализация для int” « value; }

void print(int value) { cout « “Перегрузка для int” « value; }

*8. Перечислите основные свойства параметризованных классов.*

Компонентные функции параметризованного класса автоматически являются параметризованными.  
Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по шаблону.  
Если дружественная функция содержит в своем описании параметр типа параметризованного класса, то каждый класс, организованный по шаблону имеет собственную параметризованную дружественную функцию.  
В параметризованном классе нельзя определить дружественные параметризованные классы.  
Шаблоны могут быть базовыми классами. Производными классами от такого класса могут быть обычные или шаблонные классы. Шаблоны могут наследоваться как от обычных, так и от шаблонных классов.  
Шаблоны функций-членов нельзя описывать как виртуальные.  
Локальные (вложенные) классы не могут содержать шаблоны в качестве своих элементов  
Определённые пользователем имена в описании шаблона рассматриваются как идентификаторы переменных. Чтобы имя рассматривалось как идентификатор типа, оно должно быть определено через ключевое слово typename

*9. Все ли компонентные функции параметризованного класса являются параметризованными?*

Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью template.

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

Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными функциями, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по данному шаблону.

*11. Могут ли шаблоны классов содержать виртуальные компонентные функции?*

Шаблоны функций, которые являются членами классов, нельзя описывать как virtual (Когда функция является членом класса, она уже автоматически является неявно виртуальной. Это означает, что при вызове функции через указатель или ссылку на базовый класс, будет вызвана соответствующая функция из производного класса, если она переопределена. Поэтому использование ключевого слова virtual в этом случае будет избыточным и не имеет смысла).

*12. Как определяются компонентные функции параметризованных классов вне определения шаблона класса*

Реализация компонентной функции шаблона класса, которая находится вне определения шаблона класса, должна включать дополнительно следующие два элемента:  
  
Определение должно начинаться с ключевого слова template, за которым следует такой же список\_параметров\_типов в угловых скобках, какой указан в определении шаблона класса.  
  
За именем\_класса, предшествующим операции области видимости (::), должен следовать список\_имен\_параметров шаблона.  
  
template<список\_типов>тип\_возвр\_значения имя\_класса<список\_  
  
имен\_ параметров> : : имя\_функции(список\_параметров){ . . . }

*Пример:*

// опрeделение функций

template <class T>

Vector<T>::Vector(int s, T k)

{

size = s;

data = new T[size];

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

data[i] = k;

}

*13. Что такое инстанцирование шаблона?*

После того как шаблон класса определён, он может использоваться для определения конкретных классов. Процесс генерации компилятором определения конкретного класса по шаблону класса и параметрам шаблона называется инстанцированием шаблона.

*Пример:*

Неявное инстанцирование:

template <typename T>

T sqrt(T x) { return x \* x; }

Если в программе вызов функции осуществляется как sqrt(-1), то компилятор формирует определение функции int sqrt(int x)

Явное инстанцирование:

template <typename T>

T sqrt(T x) { return x \* x; }

template double sqrt(double);

template int sqrt(int);

Через явное инстанцирование будут доступны только те типы, которые были явно инстанцированы ключевым словом template.

*14. На каком этапе происходит генерирование определения класса по шаблону?*

При включении шаблона класса в программу никакие классы на самом деле не генерируются до тех пор, пока не будет создан экземпляр шаблонного класса, в котором вместо параметра шаблона указывается конкретный тип. Экземпляр создается либо объявлением объекта, либо объявлением указателя на инстанцированный шаблонный тип с присваиванием ему адреса с помощью операции new.

*Пример:*

Vector <int> l(23, 17);

Vector <float>\* pl = new Vector<float>(19.1, 0.95);

Встретив такие объявления, компилятор генерирует код исходного класса.  
В проекте, состоящем из нескольких файлов, определение шаблона класса обычно выносится в отдельный файл. Но для того, чтобы инстанцировался конкретный экземпляр шаблона класса необходимо, чтобы определение шаблона находилось в одной единице трансляции с этим экземпляром. Поэтому все определение шаблонного класса размещается в заголовочном файле.