МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра информационные технологии и автоматизированные системы

**Дисциплина Информатика**

**Шаблоны классов**

Выполнил студент ИВТ-22-2б:

Мифтахов Марат Ринатович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пермь, 2023

**Постановка задачи**

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

**Задание**

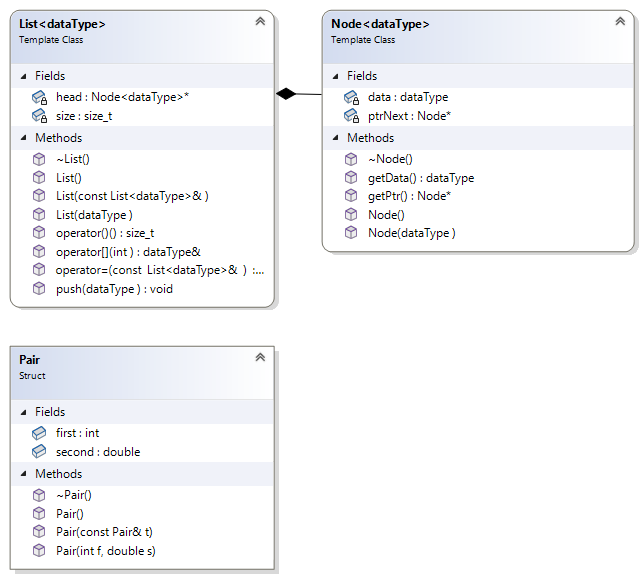
1. Класс-контейнер Список
2. Поля типа int

Операции:

1. [] - доступ по индексу
2. () - определение размера списка
3. вектор — уможение элементов списков a[i]b[i]

**Диаграмма классов**

Ниже представлена UML-диаграмма классов.



**Программный код**

Файл list.h:

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

template <class dataType>

class List;

template <class dataType>

class Node;

template <class dataType>

class Node

{

friend class List<dataType>;

public:

Node();

Node(dataType);

~Node();

dataType getData();

Node \*getPtr();

private:

dataType data;

Node \*ptrNext;

};

template <class dataType>

class List

{

public:

List();

List(dataType);

List(const List<dataType> &);

~List();

dataType &operator[](int);

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

size\_t operator()();

friend List<dataType> &operator\*(List<dataType>& sList, List<dataType>& fList)

{

List<dataType> tmpList;

int maxSize = fList.size > sList.size ? fList.size : sList.size;

int minSize = fList.size <= sList.size ? fList.size : sList.size;

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

{

dataType a = fList[i];

dataType b = sList[i];

tmpList.push(a \* b);

}

return tmpList;

}

friend std::ostream &operator<<(std::ostream &out, const List<dataType> &tList)

{

Node<dataType> \*curr = tList.head;

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

{

out << curr->getData() << " ";

curr = curr->getPtr();

}

return out;

}

friend std::istream &operator>>(std::istream & in, List<dataType> & tList)

{

dataType a;

in >> a;

tList.push(a);

return in;

}

void push(dataType);

private:

Node<dataType> \*head;

size\_t size;

};

template <class dataType>

Node<dataType>::Node()

{

// data = 0;

ptrNext = nullptr;

}

template <class dataType>

Node<dataType>::Node(dataType data)

{

this->data = data;

ptrNext = nullptr;

}

template <class dataType>

Node<dataType>::~Node()

{

if (ptrNext != nullptr)

{

delete ptrNext;

}

}

template <class dataType>

dataType Node<dataType>::getData()

{

return data;

}

template <class dataType>

Node<dataType> \*Node<dataType>::getPtr()

{

return ptrNext;

}

/// <summary>

/// Node\* head;

/// </summary>

template <class dataType>

List<dataType>::List()

{

head = nullptr;

size = 0;

}

template <class dataType>

List<dataType>::List(dataType data)

{

head = new Node<dataType>(data);

size = 1;

}

template <class dataType>

List<dataType>::List(const List<dataType> &tList)

{

Node<dataType> \*curr = tList.head;

while (curr != nullptr)

{

dataType a = curr->data;

this->push(a);

curr = curr->ptrNext;

}

}

template <class dataType>

List<dataType>::~List()

{

if (head != nullptr)

{

delete head;

head = head->ptrNext;

}

size = 0;

}

template <class dataType>

void List<dataType>::push(dataType data)

{

if (head == nullptr)

{

head = new Node<dataType>(data);

}

else

{

Node<dataType> \*curr = head;

while (curr->ptrNext != nullptr)

{

curr = curr->ptrNext;

}

curr->ptrNext = new Node<dataType>(data);

}

size++;

}

template <class dataType>

dataType &List<dataType>::operator[](int indx)

{

Node<dataType> \*curr = head;

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

{

curr = curr->ptrNext;

}

return curr->data;

}

template <class dataType>

size\_t List<dataType>::operator()()

{

return size;

}

template <class dataType>

List<dataType> &List<dataType>::operator=(const List<dataType> &tList)

{

Node<dataType> \*curr = tList.head;

while (curr != nullptr)

{

dataType a = curr->data;

this->push(a);

curr = curr->ptrNext;

}

return \*this;

}

Файл main.cpp:

#include "Header.h"

#include "Pair.h"

#include <iostream>

using namespace std;

int main() {

List<int> listInt;

List<Pair> listPair;

Pair c;

listPair.push(c);

c.first = 3;

c.second = 5.5;

listPair.push(c);

cout << "listPair: " << listPair << endl;

cout << "Enter first elem in da listPair" << endl;

cin >> listPair[1];

cout << "listPair[1] " << listPair[1] << endl;

listInt.push(3);

listInt.push(5);

cout <<"listInt: " << listInt << endl;

List<double> listDouble;

listDouble.push(4.5);

listDouble.push(6.4);

cout << "listDouble: " << listDouble << endl;

cout << "Enter elements in da list double" << endl;

cin >> listDouble;

cout << listDouble;

return 0;

}

void listFillFunction(List& tList, int n) {

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

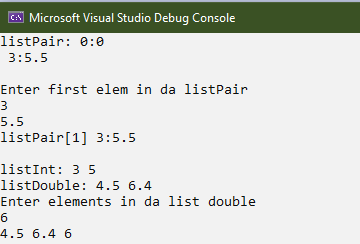
cin >> tList;

}

}

**Вывод программы**

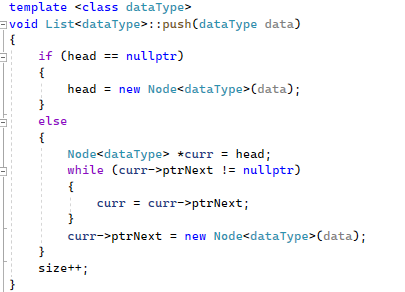
Ниже представлен вывод программы на консоль.

****

**Контрольные вопросы**

1. В чем смысл использования шаблонов?

Ответ: в создании классов или функций с разными типами данных



1. Каковы синтаксис/семантика шаблонов функций?

Ответ: template<параметры> возвращаемое значение имя (параметры) {...}

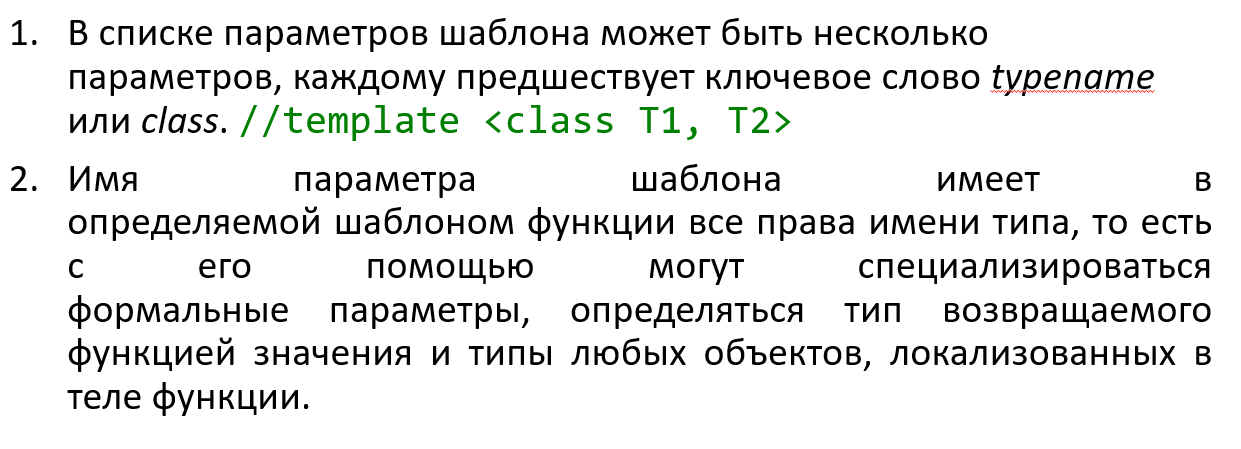
1. Каковы синтаксис/семантика шаблонов классов?

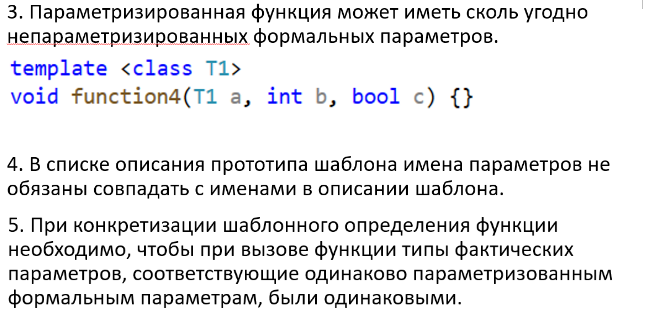
Ответ: template<параметры> class имя {...}

1. Что такое параметры шаблона функции?

Ответ: параметры шаблона функции это типы данных, которые будут передаваться в функцию

1. Перечислите основные свойства параметров шаблона функции.

Ответ: 



1. Как записывать параметр шаблона?

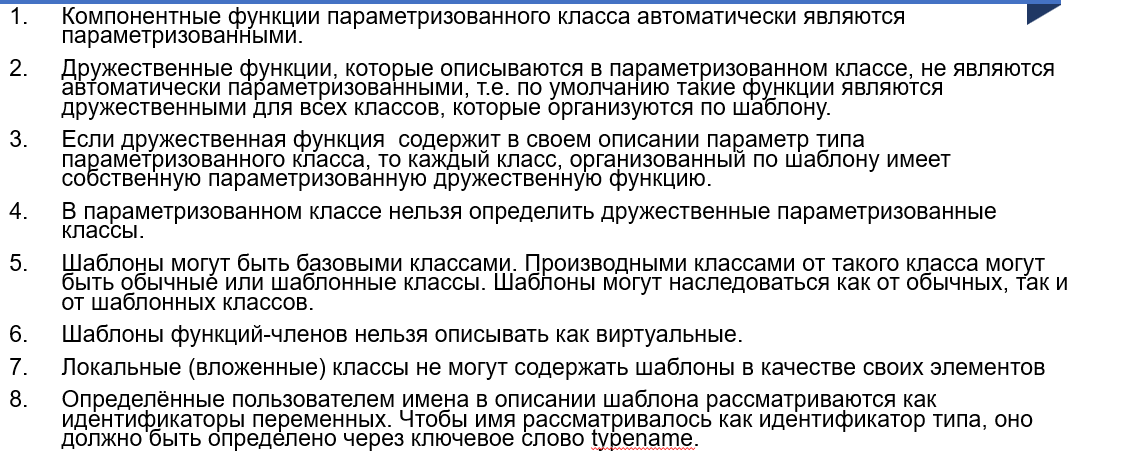
Ответ: template <class/typename TYPE>

1. Можно ли перегружать параметризованные функции?

Ответ: Да

1. Перечислите основные свойства параметризованных классов.

Ответ:



1. Все ли компонентные функции параметризованного класса являются параметризованными?

Ответ: Да

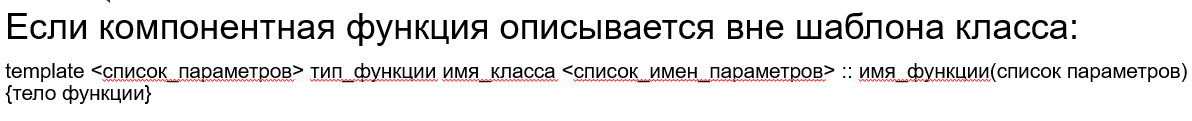
1. Являются ли дружественные функции, описанные в параметризованном  
   классе, параметризованными?

Ответ: Нет

1. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Ответ: Нет

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

Ответ: 

1. Что такое инстанцирование шаблона?

Ответ: Процесс генерации компилятором конкретного класса по шаблону при создании экземпляра параметризованного класса.

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

Ответ: На этапе создания шаблона класса