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

ПНИПУ

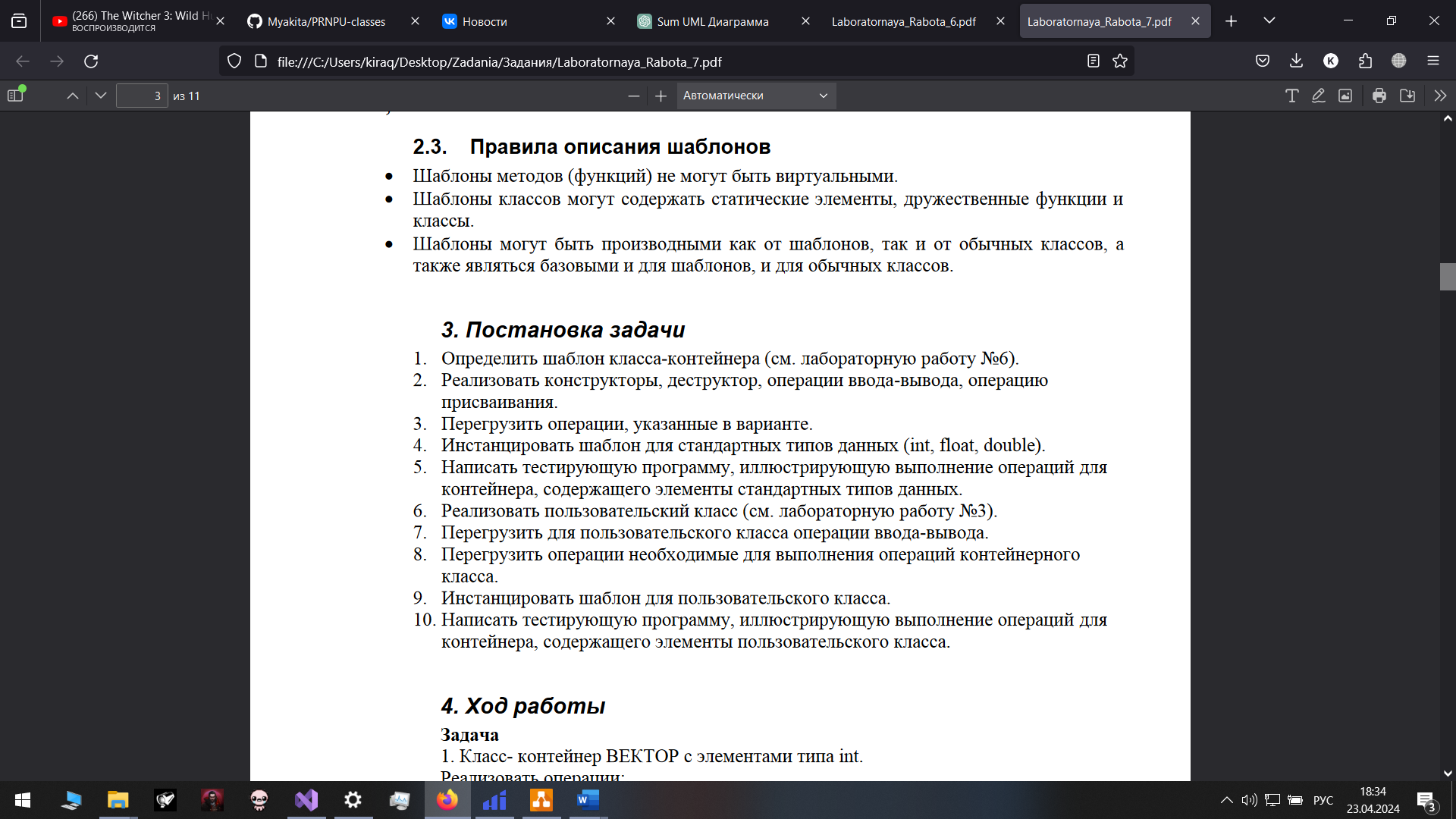
Лабораторная работа  
«Виртуальные функции. Наследование»

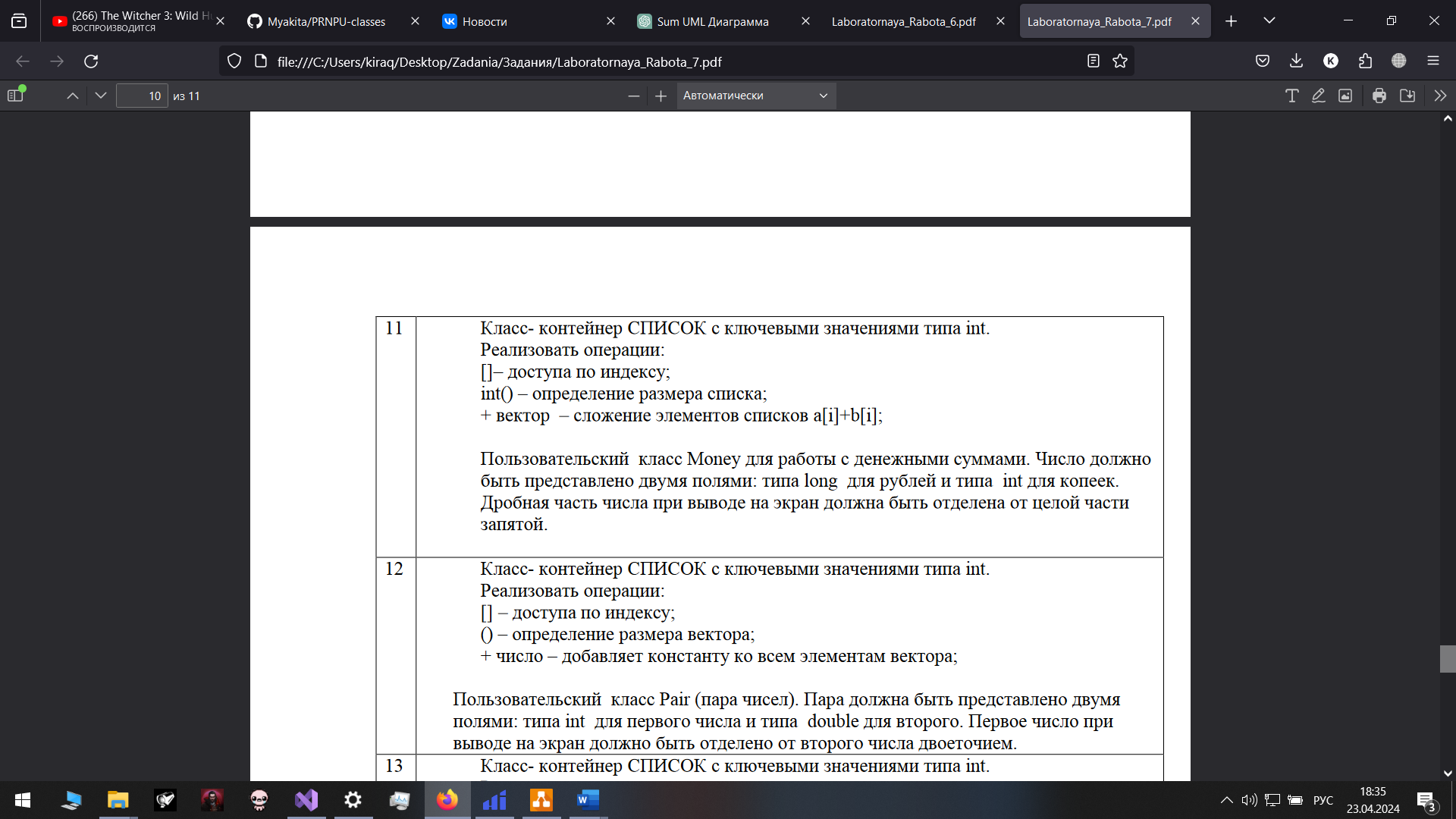
Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

Постановка задачи:





Анализ задачи:

1. **Шаблонный класс Container:** Создан шаблонный класс Container, который представляет собой обобщенный контейнер для хранения элементов. Он имеет методы для ввода, вывода и присваивания элементов.
2. **Класс Money:** Создан класс Money, представляющий деньги в виде рублей и копеек. В классе определены методы для ввода и вывода значений денежных сумм.
3. **Перегрузка операторов для класса Money:** Определены операторы << и >> для вывода и ввода денежных сумм.
4. **Перегрузка оператора присваивания для контейнера Money:** В классе Container перегружен оператор присваивания для работы с классом Money, что позволяет корректно копировать и присваивать элементы контейнера Money.
5. **Основная функция main:** В основной функции создаются экземпляры контейнеров различных типов (int, double, Money), для которых производится ввод элементов, а затем вывод на экран.

Код:

#include <iostream>

using namespace std;

template <typename T>

class Container {

private:

T\* array;

int size;

public:

Container(int s);

Container(const Container<T>& other);

~Container();

void input();

void output();

Container<T>& operator=(const Container<T>& other);

};

template <typename T>

Container<T>::Container(int s) : size(s) {

array = new T[size];

}

template <typename T>

Container<T>::Container(const Container<T>& other) : size(other.size) {

array = new T[size];

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

array[i] = other.array[i];

}

}

template <typename T>

Container<T>::~Container() {

delete[] array;

}

template <typename T>

void Container<T>::input() {

cout << "Enter " << size << " elements:" << endl;

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

cin >> array[i];

}

}

template <typename T>

void Container<T>::output() {

cout << "Container elements:" << endl;

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

cout << array[i] << " ";

}

cout << endl;

}

template <typename T>

Container<T>& Container<T>::operator=(const Container<T>& other) {

if (this == &other) {

return \*this;

}

delete[] array;

size = other.size;

array = new T[size];

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

array[i] = other.array[i];

}

return \*this;

}

class Money {

friend ostream& operator<<(ostream& os, const Money& money);

friend istream& operator>>(istream& is, Money& money);

private:

long rubles;

int kopecks;

public:

Money(long r = 0, int k = 0) : rubles(r), kopecks(k) {}

void input() {

cout << "Enter rubles: ";

cin >> rubles;

cout << "Enter kopecks: ";

cin >> kopecks;

}

void output() const {

cout << rubles << "," << kopecks << endl;

}

};

ostream& operator<<(ostream& os, const Money& money) {

os << money.rubles << "," << money.kopecks;

return os;

}

istream& operator>>(istream& is, Money& money) {

cout << "Enter rubles: ";

is >> money.rubles;

cout << "Enter kopecks: ";

is >> money.kopecks;

return is;

}

Container<Money>& Container<Money>::operator=(const Container<Money>& other) {

if (this == &other) {

return \*this;

}

delete[] array;

size = other.size;

array = new Money[size];

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

array[i] = other.array[i];

}

return \*this;

}

int main() {

Container<int> intContainer(5);

intContainer.input();

intContainer.output();

Container<double> doubleContainer(3);

doubleContainer.input();

doubleContainer.output();

Container<Money> moneyContainer(2);

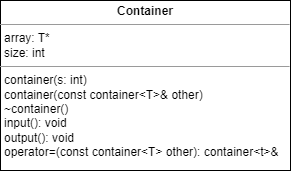
moneyContainer.input();

moneyContainer.output();

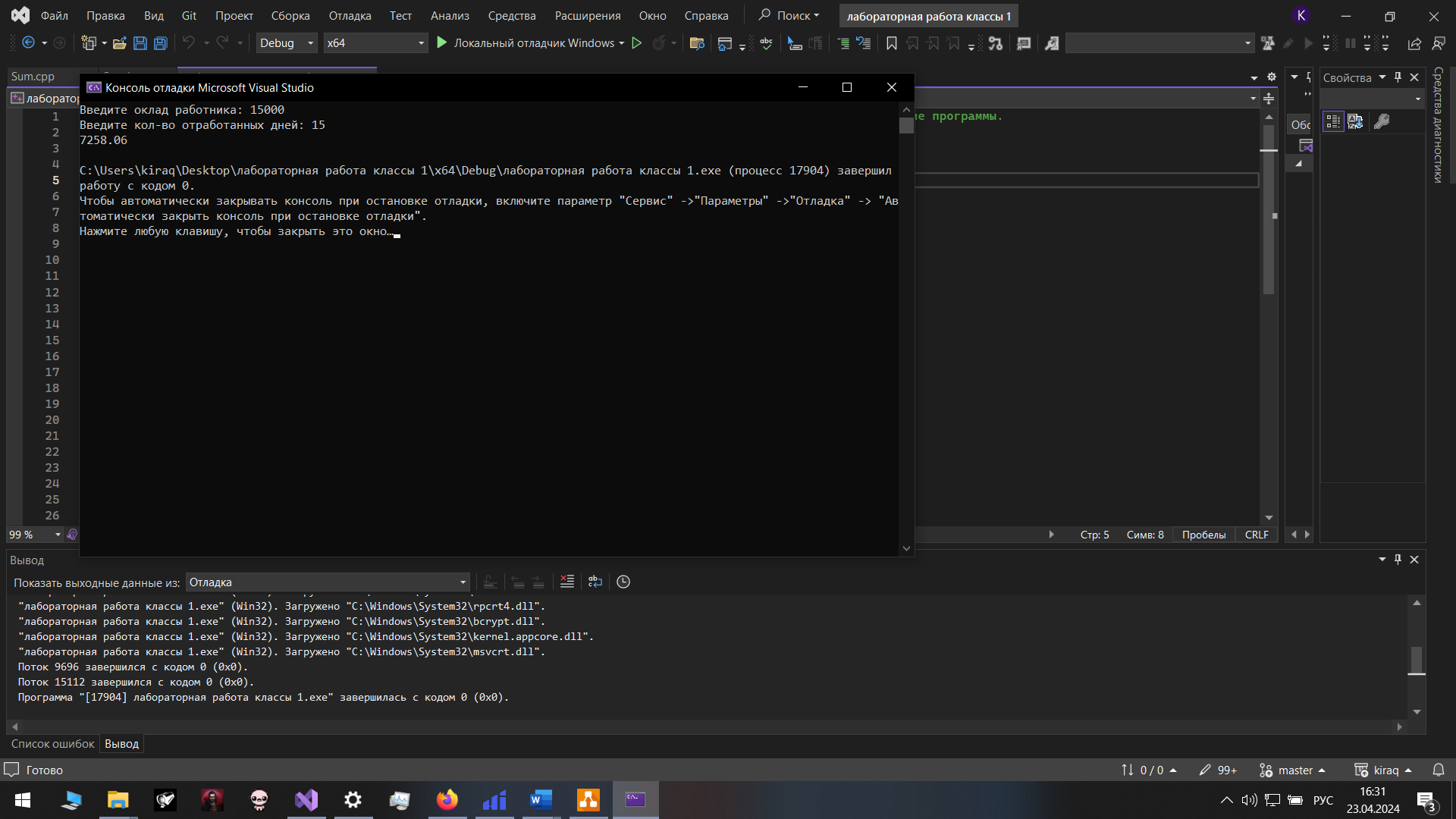
return 0;

}

UML Диаграммы:



Вывод:



программа работает корректно и выдаёт требуемый результат.  
Ответы на вопросы:

 **Смысл использования шаблонов:** Шаблоны позволяют писать обобщенный код, который может работать с различными типами данных, без необходимости дублирования кода для каждого конкретного типа. Это увеличивает гибкость, повышает переиспользование кода и облегчает разработку, так как позволяет писать универсальный код, который может быть применен к разным типам данных.

 **Синтаксис/семантика шаблонов функций:**

* **Синтаксис:** Объявление шаблона функции начинается с ключевого слова template, за которым следует список параметров шаблона в угловых скобках < >, а затем обычное объявление функции.
* **Семантика:** Шаблон функции позволяет определить функцию, которая может работать с различными типами данных. Параметры шаблона функции указываются в угловых скобках и могут быть использованы в теле функции для описания алгоритма.

 **Синтаксис/семантика шаблонов классов:**

* **Синтаксис:** Объявление шаблона класса аналогично шаблону функции, но с ключевым словом template перед объявлением класса.
* **Семантика:** Шаблон класса позволяет определить обобщенный тип данных, который может работать с различными типами данных внутри класса. Параметры шаблона класса могут использоваться для определения типов членов класса, методов и конструкторов.

 **Параметры шаблона функции:** Параметры шаблона функции - это обобщенные типы данных или значения, которые передаются в шаблон функции во время его использования для конкретизации типов данных или значений внутри функции.

 **Основные свойства параметров шаблона функции:**

* Могут быть типами данных или значениями.
* Могут быть константными или неконстантными.
* Могут быть обязательными или необязательными.
* Могут быть одиночными или множественными.

 **Запись параметра шаблона:** Параметры шаблона функции записываются в угловых скобках < > после ключевого слова template и могут быть любыми допустимыми идентификаторами. Например:

 template <typename T>

void myFunction(T parameter) {

// код функции

}

 **Перегрузка параметризованных функций:** Да, шаблонные функции могут быть перегружены. Можно иметь несколько шаблонных функций с различными параметрами шаблона или с различным числом параметров.

 **Основные свойства параметризованных классов:**

* Могут содержать шаблонные методы и/или шаблонные данные.
* Могут быть параметризованы типами данных или значениями.
* Могут быть полностью или частично специализированы.

 **Все ли компонентные функции параметризованного класса являются параметризованными:** Нет, не все. В зависимости от реализации, часть функций может быть обычными, не зависящими от параметров шаблона.

 **Дружественные функции в параметризованных классах:** Да, дружественные функции, описанные в параметризованных классах, также являются параметризованными и могут использовать параметры шаблона.

 **Виртуальные компонентные функции в шаблонах классов:** Да, шаблоны классов могут содержать виртуальные компонентные функции.

 **Определение компонентных функций параметризованных классов вне определения шаблона класса:** Компонентные функции параметризованных классов могут быть определены как внутри, так и вне определения шаблона класса. Определение функций за пределами шаблона класса может быть полезно для улучшения читаемости кода и разделения интерфейса и реализации.

 **Инстанцирование шаблона:** Инстанцирование шаблона - это процесс создания конкретной реализации шаблона функции или класса для определенного набора параметров.

 **Генерирование определения класса по шаблону:** Генерация определения класса по шаблону происходит во время компиляции, когда шаблон используется с конкретными параметрами. Компилятор создает экземпляр класса, заменяя параметры шаблона на конкретные типы данных или значения.