



ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
УПРАВЛЕНИЕ ЦИФРОВЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

Кафедра «Информационные технологии»

Сборник упражнений по дисциплине

«Объектно-ориентированное программирование»

Авторы
Рашидова Е.В.,
Зубарева Е.Г.

Ростов-на-Дону, 2023

Аннотация

Методические указания и задания для лабораторных работ №1-7 по курсу «Объектно-ориентированное программирование».

Авторы

к.ф.-м.н. доц. Рашидова Е.В.,
ст. преп. Зубарева Е.Г.





Оглавление

Лабораторная работа №1 Тема: "Классы. Конструкторы и деструкторы"	4
Лабораторная работа №2 Тема: "Друзья класса"	5
Лабораторная работа №3 Тема: "Перегрузка операторов"	8
Лабораторная работа №4 Тема: "Исследования механизма единого наследования"	11
Лабораторная работа №5.	13
Часть 1 Тема работы: Исследование механизма множественного наследования	13
Часть 2 Тема: Механизм виртуальных функций и его применение в программных проектах.	15
Лабораторная работа №6 Тема: "Манипуляторы и управление потоками ввода/вывода"	17
Лабораторная работа №7 Тема: "Шаблоны функций и шаблоны классов"	20

ЛАБОРАТОРНАЯ РАБОТА №1

ТЕМА: "КЛАССЫ. КОНСТРУКТОРЫ И ДЕСТРУКТОРЫ"

1. Создать структуру Student ("Студент") содержащую следующие поля:

- имя студента
- отчество студента
- фамилию студента
- год рождения
- группа
- средний балл успеваемости.

2. Определить конструктор для инициализации полей структуры со значениями по умолчанию. Определить деструктор. Написать тестовый пример.

3. Изменить в описании структуры ключевое слово struct на class.

Запустить программу. Какие возникли проблемы? Почему? Как их исправить?

4. Написать интерфейсные функции доступа к полям класса (получить/задать значение поля).

5. Внести в конструкторы и деструктор выдачу сообщений на экран о том, какая функция была вызвана. Модифицировать функцию main следующим образом:

```
void main(void)
{
    cout<<"Вход в функцию main()"<<endl;
    ...
    <тело_main()>
    ...
    cout<<"Выход из функции main()"<<endl;
}
```

Выяснить время вызовов конструкторов и деструкторов.

7. Описать глобальную функцию Student test(Student s){return s;}

Вызвать ее в основной программе. Что произошло и почему?

8. Изменить передачу параметра функции test на передачу по ссылке. Что изменилось?

9. Изменить возврат результата функции test на передачу по ссылке. Что изменилось?

ЛАБОРАТОРНАЯ РАБОТА №2 ТЕМА: ДРУЗЬЯ КЛАССА"

Друг класса - это функция, которая не является членом этого класса, но которой доступны члены класса из закрытого (private) и защищенного (protected) разделов класса.

Друг класса, как и функции-члены класса, является частью интерфейса класса. Поэтому объявление друга класса должно быть включено в описание этого класса, и более того, оно может появиться только в описании класса.

Таким образом, привилегии функций-друзей определяются разработчиком класса, а не пользователем класса. Это исключает возможность появления "непрощенных друзей" и не приводит к разрушению механизма защиты.

Для объявления функции-друга нужно в описание класса поместить ее прототип, перед которым расположить ключевое слово friend. Например:

```
friend int foo(int, char*);
```

Являясь частью интерфейса класса, функция-друг класса тем не менее не является членом класса. Из этого вытекают несколько следствий.

Во-первых, неважно, в каком из разделов описания класса (private, protected или public) расположить объявления друзей. Обычно принято описания друзей группировать сразу после заголовка класса.

Во-вторых, в отличие от функций-членов, функции-другу при вызове не передается указатель this. Это значит, что при вызове функции-друга нужно явно указывать ей объект, для которого она вызывается, а обычные механизмы доступа к полям класса (через точку и "стрелочку") теряют смысл.

ПРИМЕР.

Рассмотрим две функции: member_set (функция-член) и friend_set (функция-друг). Они используются для установки значения защищенного поля данных 'a' некоторого класса Test.

```
class Test{
    int a;
    friend void friend_set(Test*,int);
public:
    void member_set(int);
};
```

```
void friend_set(Test* ptr,int Value)
```



Объектно-ориентированное программирование

```
{ ptr->a=Value;} // нужно явно указать адреса объекта;  
// this здесь не передается!
```

```
void Test::member_set(int Value)  
{ a=Value;} // фактически выполняется как this-  
>a=Value;
```

```
void foo(void)  
{ // применение  
  Test x;  
  friend_set(&x,10);  
  x.member_set(10);  
}
```

Функция может быть одновременно другом нескольких классов. Это может повысить эффективность и исключить нужду в специальных функциях-членах, выполняющих ту же роль и создаваемых в каждом из классов.

Если в дружественной объявляется функция, которая используется в программе как перегружаемая, то другом становится лишь функция с конкретно заданными типами параметров. Т.е., нужно описывать как друга каждый вариант перегружаемой функции, который нужно сделать дружественным.

Чтобы сделать друзьями функции-члены некоторого класса, достаточно объявить дружественным это класс:

```
class B{  
  friend class Test; // служебное слово class можно опус-  
кать  
  ... };
```

Объявление целого класса дружественным предполагает, что все закрытые и защищенные имена (private и protected) класса, предоставляющего дружбу, могут использоваться функциями, получающими эту привилегию. Например,

```
class Test{  
  friend class B;  
  int a;};  
class B {  
  void foo(Test& t)  
  { cout << t.a ; } };
```

Объектно-ориентированное программирование

Дружественным можно объявлять класс, который еще не был описан или даже не был объявлен. Это же относится и к функциям, которые могут объявляться друзьями еще до появления их прототипа или описания.

Для того чтобы определить взаимную дружбу двух классов, нужно в каждом из них объявить дружественным другой.

Дружественность не является транзитивным свойством: если класс Y является другом класса X , а класс Z является другом класса Y , то класс Z не является другом X , если только это не объявлено явно.

Задание

1. Описать класс `Test` с защищенным числовым полем W и функцией Z , которая выводит сообщение "Это закрытая функция класса `Test`".

Написать конструктор для инициализации объектов класса `Test` с одним параметром, принимающим по умолчанию значение 1.

Объявить другом класса функцию `fun`, которая не возвращает значений и получает указатель на объект типа `Test`.

2. Описать на внешнем уровне функцию `fun`, которая выводит на экран значение параметра W и вызывает из класса `Test` функцию Z .

3. В функции `main` описать переменную класса `Test` (без явной инициализации) и применить к ней функцию `fun`.

4. Придумать и реализовать программу - пример использования двух классов A и B , в которой A друг B .



ЛАБОРАТОРНАЯ РАБОТА №3

ТЕМА: "ПЕРЕГРУЗКА ОПЕРАТОРОВ"

Перегрузка операторов напоминает перегрузку функций и является одним из видов перегрузки функций, но при этом перегружаемый оператор всегда связан с классом. Например, в классе, поддерживающем стек, оператор "+" можно перегрузить для добавления элемента в стек, а оператор "-" для выталкивания элементов из стека. Перегружаемый оператор сохраняет свое первоначальное значение, просто набор типов, к которым его можно отнести расширяется. После перегрузки операции над объектами новых классов выглядят точно так же, как операции над встроенными типами. Кроме того, перегрузка операторов лежит в основе системы ввода-вывода в языке C++.

Перегрузка операторов производится с помощью операторных функции, которые определяют действия перегружаемых операторов применительно к соответствующему классу. Операторные функции создаются с помощью ключевого слова **operator**. Операторные функции могут быть как членами класса, так и обычными функциями. Как правило, обычные операторные функции объявляются дружественными по отношению к классу, для которого они перегружают оператор.

Создание операторной функции-члена имеет следующий вид:

```
тип_возвращаемого_значения имя_класса::  
operator#(список аргументов)  
{ . . . // Операции }
```

Обычно операторная функции возвращает объект класса, с которым она работает, тем не менее, тип возвращаемого значения может быть любым. Символ # заменяется перегружаемым оператором. Например, если в классе перегружается оператор умножения "*" операторная функция-член называется **operator ***. При перегрузке унарного оператора список аргументов остается пустым. При перегрузке бинарного оператора список аргументов содержит один параметр.

Пример. Программа создает класс Комплексное число, в котором хранятся реальная и мнимая части числа и перегружается операция сложения "+".

```
#include <iostream.h>
```

```
class Complex {  
double Re,Im;
```




Объектно-ориентированное программирование

```
public:
    Complex() {}
    Complex(double a, double b){ Re=a; Im=b;}
    void show() {
        cout<<"("<<Re<<","<<Im<<")\n";
    }
    Complex operator+(Complex ob);
    Complex operator++();
};

// перегрузка "+" для класса Complex
Complex Complex::operator+(Complex ob)
{
    Complex temp;
    temp.Re=ob.Re+Re;
    temp.Im=ob.Im+Im;
    return temp;
}

// перегрузка префиксного инкремента "++" для
// класса Complex
Complex Complex::operator++()
{
    Re++;
    Im++;
    return *this; //возврат объекта, генерирующего вы-
зов
}

int main() {
    Complex ob1(10,5), ob2(7,11);
    ob1.show(); //вывод на экран (10,5)
    ob2.show(); //вывод на экран (7,11)

    ob1=ob1+ob2;
    ob1.show(); //вывод на экран (17,16)

    ob2++;
    ob2.show(); //вывод на экран (8,12)

    return 0;
}
```

Функция **operator+** имеет только один параметр, а перегружает бинарный оператор (двухместную операцию). Причина заключается в том, что операнд, стоящий в левой части оператора, передается операторной функции неявно с помощью указателя `this`. Операнд, стоящий в правой части оператора, передается через параметр `ob`. **Вывод: при перегрузке бинарного оператора вызов операторной функции генерируется объектом, стоящим в левой части оператора.**

```
Допускается следующее выражение: (ob1+ob2).show();  
//вывод на экран суммы ob1+ob2
```

В этой ситуации операторная функция создает новый объект, который уничтожается после возвращения из функции `show()`.

Замечание. При вызове функции-члена ей неявно передается указатель на вызывающий объект. Этот указатель называется `this`. Указатель `this` автоматически передается всем функциям-членам. Дружественные функции не являются функциями-членами, им не передается указатель `this`, статические функции-члены также не получают этот указатель.

Задание

1. Определите класс `date` (дата), содержащий три закрытых члена типа `int`: `day` (день), `month` (месяц), `year` (год) и массив, определяющий количество дней каждого месяца: `days[13]={0,31,28,31,30,31,30,31,31,30,31,30,31}`. Напишите конструкторы класса (Сколько конструкторов необходимо?) и функцию, показывающую дату. Перегрузите бинарные операции "+" и "-", которые выполняют следующие действия: "даты + дата", "дата - дата", изменение даты на заданное число дней: "дата + int", "int+дата" (две последние операции различны, перестановка операторов транслятором не производится), "дата-int", унарные операции "++" и "--" (переход к следующей дате, к предыдущей дате). В функции `main()` покажите работы определенных в классе перегружаемых операций.

ЛАБОРАТОРНАЯ РАБОТА №4**ТЕМА: "ИССЛЕДОВАНИЯ МЕХАНИЗМА ЕДИНОГО НАСЛЕДОВАНИЯ"**

Цели работы:

1. создание простой иерархии классов и изучение методов инициализации объектов производных классов;
2. исследование конструкции объектов производных классов;
3. исследование уровней защищенности унаследованных компонент базового класса в объектах производного класса;
4. исследование методов доступа к одноименным функциям базового и производных классов.

Задание

1. Создайте базовый класс `BASE`, в котором опишите
 - в разделе `public` поле `int i` ;
 - в разделе `protected` поле `long l` ;
 - в разделе `private` поле `double d`.Напишите конструктор, инициализирующий поля `i`, `l` и `d` тремя задаваемыми значениями.
Объясните различия и сходства между закрытыми (`private`) и защищенными (`protected`) членами класса.
2. Создайте класс `DERIVED`, производный от класса `Base` (наследование типа `public`), в котором в разделе `private` опишите поле `float f`.
Напишите конструкторы класса `DERIVED`:
 - конструктор без параметров;
 - конструктор с 4-мя параметрами для инициализации всех полей объекта.
3. В функции `main` описать неинициализированный объект класса `DERIVED` и откомпилировать программу. Если возникнут проблемы, устранить их. Вывести размеры типов `BASE` и `DERIVED` и объяснить результаты.
4. Описать инициализированный объект класса `Derived`. Продемонстрировать, инициализацию каких полей, унаследованных от класса `Base`, можно выполнять с помощью присваивания непосредственно в конструкторе класса `Derived`. Какие поля класса `Derived` обязательно нужно инициализировать с помощью конструктора класса `Base`? Для исследования можно вносить необхо-

Объектно-ориентированное программирование

димые изменения в конструкторы классов Base и Derived.

5. Перегрузить операцию вставки в поток для объектов класса Derived таким образом, чтобы выводились адреса и значения всех полей объекта. К каким полям, унаследованным от класса Base, нет доступа? Для снятия проблемы добавить в классе Base необходимые интерфейсные функции. Создав объект класса Derived, исследовать размещение полей в памяти. Привести схематическую структуру объекта.

6. Описать класс Derived_1, производный (public) от класса Derived и не имеющий новых полей. В классе описать конструктор со всеми необходимыми параметрами (сколько их нужно?).

Класс имеет общедоступную функцию void foo(), которая модифицирует значения полей, унаследованных от базового класса (i++; l+=1;). Откомпилировать программу. Заменить тип наследования Derived от Base на private и вновь откомпилировать программу. Какая возникла проблема? Для ее решения использовать возможность восстановления уровня доступа к компонентам базового класса.

7. Вернуть для Derived тип наследования public. На глобальном уровне и в классах Base и Derived описать функции void ff(), которые сообщают о своей принадлежности к классу или глобальному уровню. В функции foo класса Derived_1 добавить вызовы всех трех функций ff. В каких разделах классов Base и Derived нужно описать функции ff, чтобы они были доступны в Derived_1? Проверить работу программы, вызвав функцию foo для какого-либо объекта класса Derived_1.

8. Оставить в функции Derived::foo только один вызов в виде ff(); и проверить работу программы в следующих вариантах. Вначале функция ff определена в классах Derived, Base и на глобальном уровне. Затем ее описание убираем вначале из класса Derived, а затем из классов Derived и Base. Как в каждом случае это отражается на работе программы?

ЛАБОРАТОРНАЯ РАБОТА №5.

Часть 1

Тема работы: Исследование механизма множественного наследования

Перед выполнением лабораторной работы изучить тему множественного наследования. Обратить внимание на проблемы множественного наследования и на виртуальные базовые классы. Все задания сопроводить пояснениями. Каждое задание должно быть реализовано в виде одного .cpp файла.

Задание 1:

1. Класс Derived является производным (public) от классов Base1 и Base2. Каждый из трех классов имеет по два конструктора, которые (кроме описанных ниже действий) выводят сообщение о вызове типа: "Конструктор Base без параметров".

2. Класс Base1, имеет одно закрытое поле i целого типа. Первый конструктор не имеет параметров и обнуляет i. Второй имеет один параметр типа int, используемый для инициализации i произвольными значениями. Класс имеет две общедоступные интерфейсные функции void put(int) и int get(void), которые позволяют изменить или прочесть значение i.

3. Класс Base2, имеет одно закрытое поле - массив name из 20 элементов. Первый конструктор не имеет параметров и инициализирует поле name словом "Пусто". Второй имеет один параметр типа char*, используемый для инициализации name значениями символьных строк. Класс имеет две общедоступные интерфейсные функции void put(char*) и char* get(void), которые позволяют изменить или прочесть значение name.

4. Класс Derived имеет одно закрытое поле ch типа char. Первый конструктор не имеет параметров и присваивает ch значение 'V' (от void – пустой). Второй конструктор имеет три параметра типов char, char* и int, используемые для инициализации соответственно полей ch, name и i. Класс имеет две общедоступные интерфейсные функции void put(char) и char get(void), которые позволяют изменить или прочесть значение ch. Кроме того, в нем объявляется как дружественная операция вставки в поток вывода, которая выводит на экран значения i, name и ch. Каждый из трех классов имеет по два конструктора, которые (кроме описанных ниже действий) выводят сообщение о вызове типа: "Конструктор Base без параметров".

5. В функции main описать переменную типа Derived без

Объектно-ориентированное программирование

инициализации и вывести ее значение с помощью перегруженной операции вставки в поток. Выяснить порядок вызова конструкторов.

6. Описать другую переменную класса `Derived`, инициализировав ее явно некоторыми значениями. Вывести значение этой переменной на экран и проанализировать порядок вызова конструкторов.

7. В конструкторе класса `Derived` с параметрами изменить порядок вызова конструкторов базовых классов. Проверить, как это отразилось на работе программы и почему.

8. Изменить порядок наследования базовых классов в описании класса `Derived` и проверить, как это отразилось на работе программы.

Задание 2:

1. Задан базовый класс `DomesticAnimal` (домашнее животное), в котором определены три защищенных поля `weight` (вес), `price` (цена) и `color` (окраска). Класс снабжен конструктором без параметров и конструктором с тремя параметрами для инициализации трех полей класса. Кроме того, определена функция `print`, выводящая значения полей и сообщение о принадлежности функции к классу `DomesticAnimal`.

2. Производными от этого класса (`public`) являются классы `Cow` (корова) и `Buffalo` (бык), в которых не определено новых полей.

3. Класс `Beefalo` (теленок) является производным (`public`) от `Cow` и `Buffalo`. Его конструктор инициализирует поля `weight`, `price` и `color` без передачи параметров своим базовым классам `Cow` и `Buffalo`.

4. Классы `Cow`, `Buffalo` и `Beefalo` имеют свои функции `print`, которые выводят сообщения о своей принадлежности к конкретному классу и выводят значения трех полей с помощью вызова `print` из `DomesticAnimal`.

5. Выявить и объяснить ошибки при компиляции, исправить программу.

6. В функции `main` описать переменные типа `Cow` и `Beefalo` (с инициализацией) и вызвать для них функцию `print`. Объяснить результаты.

Часть 2

Тема: Механизм виртуальных функций и его применение в программных проектах.

Перед выполнением лабораторной работы изучить тему виртуальных функций. Повторить процесс компиляции. Все задания сопроводить пояснениями. Каждое задание должно быть реализовано в виде одного .cpp файла.

Задание 1:

1. Рассматривается иерархия классов геометрических фигур. В качестве базового используется абстрактный класс Figure, в котором объявлены общие для всех фигур способности (виртуальные функции):

1. `double area(void)`, вычисляющая и возвращающая площадь соответствующей фигуры;
2. `void show(void)`, выводящая информацию о типе фигуры (круг, прямоугольник и т.п.), о заданных размерах фигуры (например, радиус для круга, или длины сторон прямоугольника) и величину площади фигуры.

2. Производные от Figure классы кругов (Circle) и прямоугольников (Rectangle). В классе Circle конструктор принимает один аргумент - радиус и проверяет, больше ли он нуля (при ошибке - выход из программы с соответствующим сообщением). В классе Rectangle конструктор имеет один или два аргумента - длины сторон (квадрат и прямоугольник, причем функция show должна идентифицировать квадрат). Для проведения исследований в классах фигур должны быть описаны public - функции, возвращающие адреса каждого из полей данных (радиуса для круга или каждой из сторон прямоугольника).

3. В функции main:

1. Создается произвольный набор конкретных фигур.
2. Для каждого типа фигур вычисляется и выводится на экран размер одного объекта, а также адрес этого объекта и адреса его полей данных. Проанализировать результаты и дать им объяснения.

3. Создать объект базового класса Figure и откомпилировать программу. Объяснить результат компиляции.

4. Из адресов построенных фигур создать массив указателей на базовый класс. Организовать цикл, в котором выводится (с помощью виртуальной функции show) информация о каждой фигуре из массива.

Задание 2:

1. На базе программы задания 1 организовать проект `figure` с отдельно компилируемыми файлами, имеющий следующую структуру:

1. Объявления классов `Figure`, `Circle` и `Rectangle` разместить в заголовочном файле `figure.h`.

2. Определения функций-членов вынести за пределы объявлений классов и собрать в отдельном файле `figure.cpp`.

3. Функцию `main` вынести в отдельный файл `fig_main.cpp`.

4. Откомпилировать проект и убедиться в его работоспособности.

5. Создать класс `Triangle` (треугольников), производный от класса `Figure`. Для этого создать новый заголовочный файл `new_fig.h`, подключающий файл `figure.h` и содержащий объявление класса `Triangle`. Конструктор класса `Triangle` имеет три аргумента (длины сторон). В нем осуществляется проверка того, что из заданных элементов может быть составлен треугольник: сумма двух сторон должна быть больше третьей. При ошибке - выход из программы с соответствующим сообщением.

6. Реализации функций-членов класса `Triangle` размещаются в отдельном файле `triangle.cpp`. Площадь треугольника вычисляем по формуле Герона: $S = \sqrt{p(p-a)(p-b)(p-c)}$, где p - полупериметр треугольника.

7. В функцию `main` вносятся следующие изменения. Изменяем подключаемый файл на `new_fig.h`, создаем объект класса `Triangle` и добавляем его в массиву указателей на объекты.

8. Добавляем в проект файл `triangle.cpp`, компилируем программу и проверяем ее работоспособность. В случае успеха заменяем в проекте файл `triangle.cpp` на объектный файл `triangle.obj` и проверяем работу нового проекта.



ЛАБОРАТОРНАЯ РАБОТА №6

ТЕМА: "МАНИПУЛЯТОРЫ И УПРАВЛЕНИЕ ПОТОКАМИ ВВОДА/ВЫВОДА"

Задание 1.

1. Что будет получено в результате работы фрагмента и почему?

```
int a=0, x;  
cout<<a?(x=1):(x=0);  
cout<<"\t"<<x;
```

2. Что будет выведено в следующем фрагменте программы?

```
char *s = "Это строка символов!";  
void *v = s;  
void *v1 = "Вторая строка!";  
int i = 1;  
cout<<v<<"\t"<<s<<"\t"<<(int*)s<<"\t"<<v1<<"\n";  
cout<<&i<<"\t"<<i;
```

Как сделать, чтобы были напечатаны обе символьные строки, использованные в этом фрагменте?

3. Выяснить, как далеко распространяется действие манипуляторов

oct, hex, dec, setfill, setw и setprecision: на ближайший операнд, на данную цепочку вывода или до явной отмены?

4. Выяснить, что определяет при выводе манипулятор setprecision:

число значащих цифр или число цифр после запятой?

Происходит ли

округление результата или лишние разряды обрезаются?

5. Задано число типа float. Задав ширину поля, например, в 12

символов, вывести его дважды: в обычной и научной нотации. Затем

Объектно-ориентированное программирование

вывести его еще 2 раза(каждый раз с новой строки) в
обычной

нотации с выравниванием влево и вправо и символом за-
полнителем

'.'.

Задание 2.

Составить программу для проверки работы форматирую-
щих операций

ввода-вывода функций-членов класса ios.

Задание 3.

Написать программу для проверки использования флагов
форматирования для выполнения операций ввода-
вывода.

Задание 4.

1. Создать манипулятор без параметров endp, который
подсчитывает

число выведенных строк и при заполнении страницы вы-
полняет

операцию перехода на новую страницу. Число строк на
странице

фиксировано в функции-манипуляторе.

При тестировании (вывод на экран) переход на новую
страницу

смоделировать выводом какой-либо строки, например, "--
.....".

2. Создать аналогичный манипулятор endp(n), но с пара-
метром,

задающим условие перехода на новую страницу: если
счетчик строк в

функции-манипуляторе имеет значение большее задан-
ного n, то

осуществить переход на новую страницу.

3. Создать манипулятор с двумя параметрами fendp(n,s),
где n - количество строк на странице

s - строка-приглашение, выводящееся в конце страни-
цы.



Задание 5.

Написать программу для проверки возможностей управления потоком

ввода: ограничить число вводимых в буфер символов с последующей

очисткой потока; проверить работу функций peek, putback, ignore.



ЛАБОРАТОРНАЯ РАБОТА №7

ТЕМА: "ШАБЛОНЫ ФУНКЦИЙ И ШАБЛОНЫ КЛАССОВ"

Задание 1.

1. Какие ошибки допущены в следующих объявлениях?

```
template <class T, class T> T f(T x);  
template <class T1, T2> void f(T1 x);  
template <class T> T f(int x);  
inline template <class T> T f(T x, T y);
```

2. Написать тестовую программу для функции `swap` и попробовать ее вызовы с различными типами аргументов (значения переменных - числа, символы, строки).

3. Написать программу, в которой определяется шаблон для функции `max(x,y)`, возвращающей большее из значений `x` и `y`. Написать специализированную версию функции `max(char*,char*)`, возвращающую "большую" из передаваемых ей символьных строк. В каждой из функций предусмотреть вывод сообщения о том, что вызвана шаблонная или специализированная функция и вывод найденного большего. Проверить работу программы на трех примерах `max('a','1')`, `max(0,1)`, `max("Hello","World")`.