

# Тема 1. Введение в C++

## Лекция 1. Простейший пример

В 2020–2021 учебном году курс лекций, читающихся на механико-математическом факультете МГУ, проходит в дистанционном режиме. Файл с видеозаписью собственно лекции выкладывается отдельно, а здесь приводится конспект сказанного на этой лекции. Это позволит слушателям не тратить время на конспектирование, однако пояснения и обсуждения многих моментов проводится именно на лекции, и не всегда отражается в ее конспективной записи. Таким образом, для ознакомления с материалом стоит прослушать именно лекцию, а содержимое данного файла сохранит основные ключевые моменты.

По учебному плану в этом семестре запланирован экзамен, а зачета нет. Поэтому традиционно уже несколько лет оценка экзамена зависит от практической работы на семинарах. Что получится в этом семестре сказать невозможно, так как очное проведение экзамена остается под большим вопросом, но все же мы попытаемся сохранить общие принципы выставления оценки. Эта оценка складывается из трех слагаемых: работы в семестре, решение практической задачи на экзамене, и ответ на “теоретические вопросы билета”. Раньше эти слагаемые соответствовали составляли 1.5+1.5+3 максимальных балла, т.е. человек мог получить 5 при небольших дефектах ответа на билет, решения задачи, или работы в семестре. Как это будет в этом семестре, мы сказать пока не в состоянии, но общий принцип сохранится, т.е. работа в семестре будет непосредственно влиять на итоговую оценку.

Программа м лекций второго семестра состоит из двух больших разделов: схемы хранения данных и некоторые алгоритмы обработки данных. По сути это квалификационный минимум программиста — знание как представить данные в своих программах и как их обрабатывать в “стандартных” ситуациях.

Первая часть курса посвящена классическим схемам хранения данных таким как стек, очередь, дек, список, дерево, граф, множество и т.п., а вторая часть знакомит с такими же алгоритмами обработки информации (поиск в деревьях, алгоритмы на графов, сжатие данных и т.п.).

Базовым инструментом работы во 2 семестре будет язык C++, так как он дает необходимые объектно-ориентированные средства, которые оказываются в данном контексте очень удобными.

Мы не будем отводить отдельное время подробному изучению языка, а сразу будем строить требуемые реализации, разъясняя по мере необходимости используемые концепции и конструкции.

Однако с самого начала все же придется ввести несколько понятий.

Объект — это некоторое образование, которое обладает внутренним состоянием (набором данных) и набором функций, позволяющих это состояние модифицировать. Функции из этого набора могут быть доступны внешнему использованию (интерфейс), но также может быть закрыты от внешнего воздействия и использоваться только как внутренние служебные алгоритмы объекта.

Объектно-ориентированная программа создает некоторое количество объектов, устанавливает им начальные состояния, а далее, при помощи взаимодействия объектов приводит их к тому состоянию, которая является конечной целью. В этом плане, программирование является аналогом управленческой деятельности, когда разнообраз-

ные исполнители координируют свои действия для достижения желаемого результата.

В терминах языка C++ базовым понятием является понятие класса.

Класс — это структура, которая содержит некоторый набор данных и также набор методов (функций), которые могут выполнять работу с данными этого класса, преобразуя эти данные и тем самым меняя состояние конкретного экземпляра класса.

Также вводятся понятия прав доступа к данным и методам. Одни операции могут выполняться кем угодно, другие операции могут быть разрешены только экземплярам данного класса или кому-то еще по определенным правилам. Разграничение прав доступа составляет одну из основных концепций языка C++.

Мы не будем здесь приводить исчерпывающее формальное изложение всех правил языка C++, а ограничимся лишь некоторыми примерами, помогающими понять суть. Для более подробной информации можно обратиться к формальным описаниям языка и справочным пособиям.

Рассмотрим формализм введения классов в C++ на примере вектора.

Многие конструкции могут быть записаны в разных формах, поэтому надо иметь в виду, что дальнейшие примеры не являются единственно возможным вариантом, и об этом даны комментарии в устной лекции.

Пусть мы хотим записать решение некоторой геометрической задачи с использованием векторов  $a, b, c, d$  и некоторыми естественными операциями с ними. Например,

```
int main()
{
    double s, r;
    Vector a, b, c, d;

    a = b + c + 3*d; // линейная комбинация векторов
    r = a*b;        // скалярное произведение

    return 0;
}
```

Т.е. нам надо определить класс Vector и задать ему операции между векторами, между векторами и числами и т.п.

Определение класса может использоваться в нескольких файлах, поэтому, обычно его помещают в заголовочный файл. В нашем случае назовем его Vector.h.

После некоторых размышлений и принятия решений по поводу функциональности, описание класса может принять примерно такой вид.

```
// файл Vector.h
#include <stdio.h>

class Vector
{
private:
    double x,y,z; // координаты вектора
public:
    // конструкторы
    Vector();
```

```
Vector(double xx, double yy, double zz);
Vector(const Vector &a);          // конструктор копирования
~Vector() = default;            // деструктор

// операции, определяемые через дружественные функции
friend double operator*(const Vector &a, const Vector &b);
friend Vector operator+(const Vector &a, const Vector &b);
friend Vector operator*(double c, const Vector &a);

// операции, определяемые через методы класса
const Vector & operator=(const Vector &b);

// распечатка вектора
void Print(FILE *f);
};
```

В описании класс только объявлен, а реализация его методов обычно размещается также в отдельном файле. Назовем этот файл Vector.cpp.

```
// файл Vector.cpp
#include "Vector.h"

// реализация класса Vector и его friend функций

Vector::Vector()
{
    x = y = z = 0;
}
Vector::Vector(double xx, double yy, double zz)
{
    x = xx; y = yy; z = zz;
}
Vector::Vector(const Vector &v)
{
    x = v.x; y = v.y; z = v.z;
}
const Vector & Vector::operator=(const Vector &b)
{
    x = b.x; y = b.y; z = b.z;
    return *this;
}
void Vector::Print(FILE *f)
{
    fprintf(f, "%f %f %f\n", x, y, z);
}

double operator*(const Vector &a, const Vector &b)
{
```

```
    return a.x*b.x + a.y*b.y + a.z*b.z;
}
Vector operator+(const Vector &a, const Vector &b)
{
    return Vector(a.x+b.x, a.y+b.y, a.z+b.z);
}
Vector operator*(double c, const Vector &a)
{
    return Vector(c*a.x, c*a.y, c*a.z);
}
```

Теперь для проверки нашей реализации можно составить работоспособную программу.

```
// файл VecProg.cpp
#include "Vector.h"

// демонстрационный пример использования класса Vector

int main()
{
    double s, r;
    Vector a, b;
    Vector c(1.0, 2.0, 4.0), d(5, 3, -7);

    a = c + 3*d;
    a.Print(stdout);
    b = a + Vector(1,2,3);
    b.Print(stdout);

    r = a*b;
    s = operator*(a,b);
    fprintf(stdout,"scal prod %f %f\n", r, s);

    return 0;
}
```

### **Что осталось недосказанным, или краткий итог.**

Определение класса может быть записано разными способами. Пока нет полного понимания, что должно присутствовать обязательно, а что можно переложить на плечи компилятора, условимся, что определение класса должно содержать:

- конструктор без параметров (можно по умолчанию =default)
- другие конструкторы с параметрами (если надо по смыслу задачи)
- конструктор копирования
- деструктор (можно по умолчанию =default)
- методы класса (по смыслу задачи)
- внутренние переменные класса нужно как правило размещать в секции private

Определение операций (перегрузка операций) может реализовываться либо дружественными функциями, либо в некоторых случаях как методы класса. В последнем случае первым аргументом операции является класс, от имени которого вызывается данный метод.

Операция может быть вызвана не только записью ее знака вместе с аргументами, но так же как и функция класса (например  $a+b$  или  $a.operator+(b)$ ).

Аргументы и возвращаемые значения целесообразно задавать как (константные) ссылки с тем, чтобы лишний раз не задействовать конструктор копирования. При этом надо понимать, что возвращать ссылку можно только на объект, существующий вне контекста данной функции, поскольку локальные объекты (переменные) функции перестают существовать после завершения функции, и ссылка на них уже не будет иметь смысла.

Более подробную информацию о перегрузке операций и прочих рассмотренных вопросах можно получить из справочной литературы или поиском в сети.

Как упражнение по данной лекции предлагается доработать класс `Vector`, внося в него множество других операций и методов, чтобы получить возможность работать с векторами “как с числами”.

Работа на ЭВМ и программирование  
механико-математический факультет  
2 курс, 2 поток, 2020-2021 уч.год

Валединский Владимир Дмитриевич

сайт с материалами: [v-dinsky.info](http://v-dinsky.info)

вопросы, комментарии и т.п. по почте

[v-dinsky@yandex.ru](mailto:v-dinsky@yandex.ru)

в теме письма пишите "лекции-2"  
на это слово будет настроен фильтр,  
тобы письмо не затерялось в массе других