

Основы C++. Вебинар №2.

Длительность: 1.5 - 2 ч.



GeekBrains

Что будет на вебинаре?

- Узнаем что такое переменная и типы данных в C++.
- Узнаем о понятии - класс памяти, области действия и время жизни переменных.
- Изучим такие типы данных как массивы, структуры, объединения, битовые поля.



Создание нового проекта и отладка

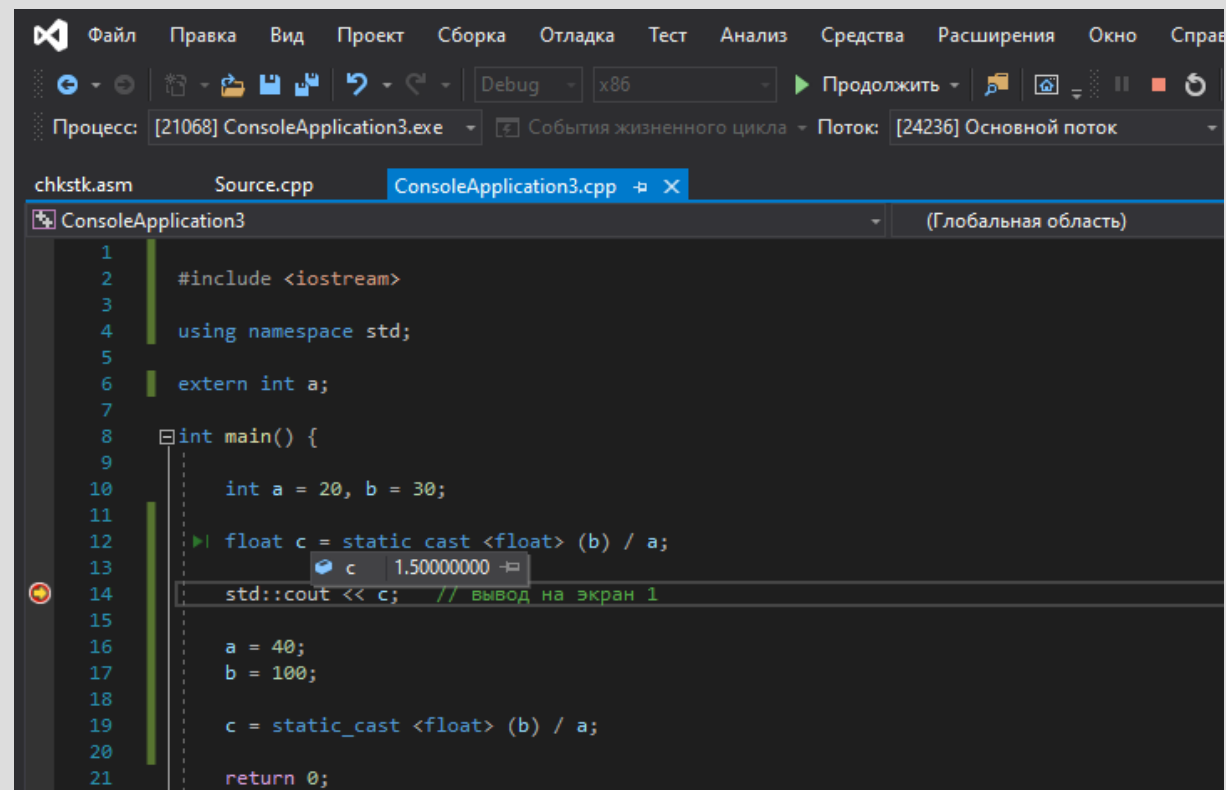
Когда вы будете программировать в своей среде программирования (IDE) создавайте — новый **консольный проект**. Этого достаточно для наших текущих потребностей.

Большинство сред позволяет делать **отладку** вашей программы — **Debugging**. Можно добавлять в программу **точки останова** — break points. В режиме отладки вы можете видеть значения переменных и выполнять программу пошагово (F10 в VisualStudio).

MS VisualStudio по умолчанию создает проекты в следующей папке (у меня):

C:\Users\Dmitry\source\repos

Еще бывает что антивирус может мешать сборке вашего проекта. Тогда его нужно временно выключать или выгружать из памяти.



```
1  #include <iostream>
2
3
4  using namespace std;
5
6  extern int a;
7
8  int main() {
9
10     int a = 20, b = 30;
11
12     float c = static_cast<float>(b) / a;
13
14     std::cout << c; // вывод на экран 1
15
16     a = 40;
17     b = 100;
18
19     c = static_cast<float>(b) / a;
20
21     return 0;
```

Переменные в C++

Каждая переменная имеет:

- **Название** — 1) нельзя начинать с цифры, 2) не должно совпадать с ключевыми словами из C++, 3) чувствительно к регистру, 4) имя должно быть уникальным.
- **Тип данных** — разные типы данных имеют разный размер занимаемой памяти;
- **Значение** — то что, хранит в себе переменная;

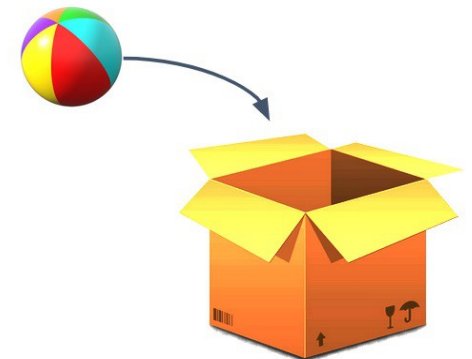
А также:

- **Класс памяти** (automatic, register, static, external).
Подробнее про классы памяти: <https://it-black.ru/klassy-pamyati-v-yazyke-si/>

```
int var1 = 100; // объявляем целочисленную переменную
                // типа int, с именем var1, и
                // инициализируем ее значением 100
```

В C++: строгая, статическая, явная типизация.

Значение переменной



Типы данных в C++

- **bool**: логический тип. Может принимать одну из двух значений true (истина) и false (ложь). Размер занимаемой памяти для этого типа точно не определен, но часто это 1 байт.
- **char**: представляет один символ в кодировке ASCII. Занимает в памяти 1 байт (8 бит). Может хранить любое значение из диапазона от -128 до 127, либо от 0 до 255.
- **short int**: представляет целое число в диапазоне от -32768 до 32767. Занимает в памяти 2 байта (16 бит).
- **int**: представляет целое число. В зависимости от архитектуры процессора может занимать 2 байта (16 бит) или 4 байта (32 бита).
- **long long**: представляет целое число в диапазоне от -9 223 372 036 854 775 808 до +9 223 372 036 854 775 807. Занимает в памяти, как правило, 8 байт (64 бита).
- **float**: представляет вещественное число ординарной точности с плавающей точкой в диапазоне +/- 3.4E-38 до 3.4E+38. В памяти занимает 4 байта (32 бита).
- **double**: представляет вещественное число двойной точности с плавающей точкой в диапазоне +/- 1.7E-308 до 1.7E+308. В памяти занимает 8 байт (64 бита).
- **void**: тип без значения

Все целочисленные переменные могут быть также **signed** и **unsigned** — то есть иметь или не иметь знак.

`unsigned int A = 4'294'967'294;` // целое число

`char C = 'F';` // один символ

`bool B = true;` // логический тип

`float D = 3.14;` // число с плавающей точкой

Инициализация переменных в C++

Правило: перед использованием переменная **всегда** должна быть инициализирована иначе в ней будет находиться мусор (случайное значение).

```
int A = 10;  
// теперь можно использовать A
```

Или

```
double B;  
...  
B = 12.13;  
// теперь можно использовать B
```

Ошибка:

```
int C = 100;  
int D, E;  
E = C + D;  
std::cout << E << std::endl;
```

// Забыли инициализировать переменную D, поэтому в E непредсказуемое значение.

Правило одного определения - One Definition Rule

ODR — (англ. One Definition Rule) - один из основных принципов языка программирования C++.

Назначение ODR состоит в том, чтобы в программе не могло появиться два или более конфликтующих между собой определения одной и той же сущности (типа данных, переменной, функции, объекта, и пр.).

Нарушения ODR зачастую не могут быть обнаружены компилятором. В большинстве случаев их обнаруживает **компоновщик (линкер)**.

Пример нарушения: наш проект состоит из двух сpp файлов в каждом из которых объявлена глобальная переменная `i`.

```
Main.cpp  
int i = 10;
```

```
Extra.cpp  
int i = 100;
```

Две функции с одинаковыми именами объявленные в 2х разных сpp файлах одного проекта — тоже будет нарушением ODR.

Спецификатор auto

Согласно стандарту C++11 можно предоставить компилятору самому выводиться тип объекта. И для этого применяется спецификатор auto. При этом если мы определяем переменную со спецификатором auto, эта переменная **должна быть обязательно инициализирована** каким-либо значением:

```
auto number = 5;
```

Полезно использовать если наш тип данных занимает много места, особенно часто при использовании библиотеки STL:

Например вместо:

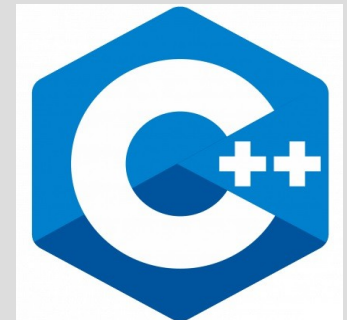
```
std::unordered_map<int, std::string>::const_iterator it = map.begin();
```

Будет:

```
auto it = map.begin();
```

Кто использовал тип auto, напишите в чат +

Подробнее о типах данных в C++: <https://metanit.com/cpp/tutorial/2.3.php>



Операция sizeof

Иногда нужно точно знать размер, который занимает та или иная переменная или объект. Точный размер зависит от вашей ОС и используемого компилятора. Это позволяет сделать операция sizeof.

```
bool flag = false;  
long long VeryLongVar = 1'000'000'000'000;  
double pi = 3.14;
```

```
std::cout << sizeof(flag) << " " << sizeof(VeryLongVar)  
<< " " << sizeof(pi) << std::endl;
```

Вывод на экран: 1 8 8

Также операцию sizeof можно применять к типам данных:

```
std::cout << sizeof(bool) << " " << sizeof(float) << std::endl;
```

Вывод на экран: 1 4

Область видимости переменных

```
/* Пример  
многострочного комментария в main.cpp */  
  
#include <iostream>  
  
int a = 100; // глобальная переменная, находится в сегменте данных  
  
int main()  
{  
    short int b = 255; // локальная переменная, находится в стеке  
    std::cout << a << " " << b;  
    return 0;  
}
```

Область видимости у переменной *a* — весь файл *main.cpp*

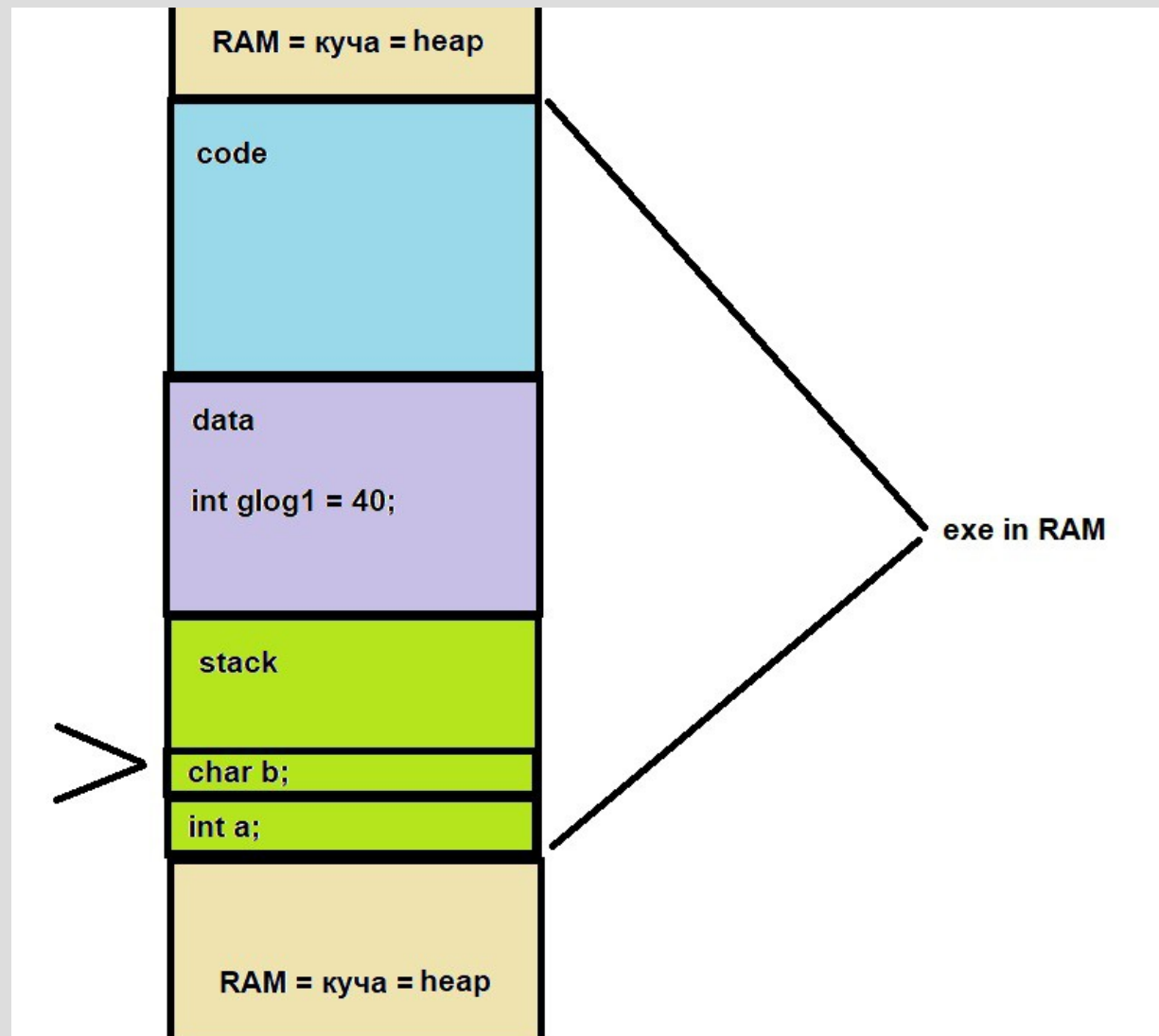
Область видимости у переменной *b* с начала ее объявления и до конца функции *main*. После выхода из функции *main* переменная *b* уничтожается!

Переменные **очень большого размера** (структуры, массивы) локально лучше не создавать так как стек может переполниться, а нужно тогда использовать **динамическое выделение памяти** (в куче) через оператор *new*.

EXE file in RAM

Программа обычно
состоит из сегментов:

- Код
- Данных
- Стека



Оператор разрешения области видимости и вывод кириллицы в консоли

```
#include <iostream>
#include <locale> // Для вызова функции setlocale

double var = 8.888;

int main()
{
    setlocale(LC_ALL, "Russian"); // подключаем русский язык в консоли

    int var = 1555;
    std::cout << "Локальная переменная: " << var << " Глобальная переменная: " << ::var;
    return 0;
}
```

Вывод на экран:

Локальная переменная: 1555 Глобальная переменная: 8.888

В случае создания двух переменных с одинаковым именем (одна из которых является глобальной, а другая локальной) при использовании в блоке, в котором была объявлена локальная переменная, можно использовать и глобальную переменную. Для ее использования нужно всего лишь применить глобальный оператор разрешения.

Глобальный оператор разрешения — это два подряд поставленные двоеточия, с помощью которых мы говорим компилятору, что хотим использовать глобальную переменную, а не локальную.

Константность в C++

Для объявления константной переменной используйте ключевое слово `const`:

```
const int A = 1000;
```

Или

```
const double B = 9.999;
```

Константа **всегда должна** быть инициализирована при объявлении и далее ее менять уже нельзя.

* Информация для продвинутых студентов:

НО, все же есть способ изменить константу с помощью оператора приведения типа `const_cast` если привести к неконстантной ссылке и менять значение через нее.

В C++ есть 4 оператора приведения (изменения) типа + один C-style cast. Они будут изучаться позже.

Псевдонимы типов: typedef и using

Ключевое слово **typedef** позволяет программисту создать псевдоним для любого типа данных и использовать его вместо фактического имени типа. Чтобы объявить typedef (использовать псевдоним типа) — используйте ключевое слово typedef вместе с типом данных, для которого создается псевдоним, а затем, собственно, сам псевдоним.

В стандарте **C++11** ключевое слово **using** может использоваться для этих же целей что и typedef.

```
typedef long CarSpeed; // создаем новый тип данных CarSpeed
```

```
int main()
{
    CarSpeed speed1 = 90;
    ...
}
```

Замена типа, **может быть полезна** если изначальный тип **очень длинный**:
Например в STL контейнерах вместо:

```
std::unordered_map<int, std::string>::const_iterator it = map.begin();
```

Будет:

```
// Создаем псевдоним длинного и странного типа данных
typedef std::unordered_map<int, std::string>::const_iterator ConstMapIter;
```

```
ConstMapIter it = map.begin(); // Теперь при объявлении можем использовать его
```

Массивы в C++

Массив — структура данных, представленная в виде группы ячеек одного типа, объединенных под одним единым именем. Массивы используются для обработки большого количества **однотипных данных**. Имя массива является указателем, их рассмотрим позже. :) Отдельная ячейка данных массива называется **элементом массива**. Элементами массива могут быть данные **любого типа**.

Пример:

```
int a[7] = {5, -12, -12, 9, 10, 0, -9};  
std::cout << "a[0] = " << a[0] << std::endl; // вывод на экран первого элемента  
a[1] = 1'000'000; // записали во второй элемента число  
std::cout << "a[6] = " << a[6] << std::endl; // вывод на экран последнего элемента
```

Размер таких классических массивов не может быть изменен динамически.
Индекс массивов в C++ всегда начинается с 0.

Пример массивов из char:

```
char Name [ ] = { 'J', 'o', 'n', '\0' }; // \0 - символ конца строки в C++  
char Name2 [ ] = "Jon"; // строка это тоже массив символов
```

```
std::cout << Name << " " << Name2 << std::endl; // выведет на экран: Jon Jon
```

```
Name[0] = 'D';  
Name2[1] = 'a';
```

```
std::cout << Name << " " << Name2 << std::endl; // выведет на экран: Don Jan
```

5	-12	-12	9	10	0	-9
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

Двумерные массивы

Двумерный массив из элементов типа int:

```
int Array[3][2] = { {1, 2}, {3, 4}, {5, 6} };
```

Можно также и так объявить:

```
int Array2[ ][2] = { {1, 2}, {3, 4}, {5, 6} };
```

И так можно:

```
const int size = 5;  
char Array3 [size][size]; // Матрица 5 x 5
```

```
// выведем на экран первый и последний элементы  
std::cout << Array[0][0] << " " << Array[2][1];  
// вывод на экран: 1 6
```

Двумерный массив из элементов типа bool:

```
bool EmptyParkingPlaces[3][3] = { {true, false, true}, {false,false,false}, {true, true, true} };  
EmptyParkingPlaces[1][1] = true;
```


Ввод информации с клавиатуры — объект cin

```
#include <iostream>

using namespace std; // используем пространство имен std во всем файле

int main() {

    cout << "Enter your name: ";
    char name [32] = { 0 };
    cin >> name;                                     // пользователь вводит свое имя

    cout << "Enter your age: ";
    unsigned int age = 0;
    cin >> age;                                       // пользователь вводит свой возраст

    cout << "Hi, " << name << "_" << age << endl;

    return 0;
}
```

Наберите такую программу дома.
Запустите ее. =)

 Консоль отладки Microsoft Visual Studio

```
Enter your name: Dima
Enter your age: 25
Hi, Dima_25
```

Перечисления в C++

Перечисление (или «перечисляемый тип») — это тип данных, где любое значение (или «перечислитель») определяется как **символьная константа**. Объявить перечисление можно с помощью ключевого слова `enum`.

Например:

```
enum Week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
```

Или

```
enum Colors  
{  
    COLOR_RED,  
    COLOR_BROWN,  
    COLOR_GRAY  
};
```

```
Colors paint = COLOR_RED;  
std::cout << paint; // Напечатается на экране 0
```

```
paint = COLOR_BROWN;  
std::cout << paint; // Напечатается на экране 1
```

Для продвинутых:

Если вам известно о таком типе `enum` как:

`enum class`

Он более безопасный чем обычный `enum` о нем можно почитать подробнее тут:

<https://ravesli.com/urok-59-klassy-enum/>

Массив из enum

Объявленный enum можно использовать при объявлении массивов,

Например:

```
#include <iostream>

// Перечисление - дни недели
enum Week { Mon, Tue, Wed, Thu, Fri, Sat, Sun };

int main()
{
    Week work_days[4] = { Mon, Wed, Fri, Sat };
    Week holi_days[3] = { Tue, Thu, Sun };

    // Работаем с нашими массивами
    std::cout << "First work day: " << work_days[0] << std::endl;
    std::cout << "First holiday: " << holi_days[0] << std::endl;
    work_days[0] = Sun;
    holi_days[2] = Mon;
    return 0;
}
```

Будет похожее задание в ДЗ

Структуры в C++

C++ позволяет программистам создавать свои собственные пользовательские типы данных, которые группируют **несколько отдельных переменных вместе**. Одним из простейших пользовательских типов данных является структура. Структура позволяет сгруппировать переменные разных типов в единое целое.

Доступ к полям структуры мы получаем через операцию точка.

```
struct Employee // Новый тип данных Сотрудник
{
    long id;           // ID сотрудника
    unsigned short age; // его возраст
    double salary;     // его зарплата
};

int main()
{
    Employee e1, e2;
    e1.id = 125037;
    e1.age = 31;
    e1.salary = 120'000.0;

    e2 = { 125038, 28, 75'000.0 };
    return 0;
}
```

Более сложные структуры

```
struct Employee { // Новый тип данных Сотрудник
    long id;          // ID сотрудника
    unsigned short age; // его возраст
    double salary;     // его зарплата
};
```

```
enum CompanySize { CS_SMALL, CS_MIDDLE, CS_BIG }; // перечисление — размер компании
```

```
struct Company { // Новый тип данных Компания
    Employee people[30]; // Ее сотрудники (30 максимум)
    Employee director;   // Директор
    CompanySize size;    // Размер компании
    unsigned int PeopleNumber; // количество сотрудников
};
```

```
int main() {
    Company comp;
    comp.director = { 125093, 45, 350'000.0 };
    comp.size = CS_MIDDLE;
    comp.PeopleNumber = 215;
    comp.people[0] = comp.director;
    comp.people[1] = { 134578, 34, 60'000.0 };

    return 0;
}
```

Правило:

Объявление структуры, перечисления или класса обязано заканчиваться точкой с запятой:

```
struct SomeStruct {
    ...
};
```

```
enum SomeEnum { ... };
```

Массив из структур

```
struct Employee { // Новый тип данных Сотрудник
    long id;           // ID сотрудника
    unsigned short age; // его возраст
    double salary;      // его зарплата
};

enum CompanySize { CS_SMALL, CS_MIDDLE, CS_BIG }; // перечисление — размер компании

struct Company { // Новый тип данных Компания
    Employee people[30]; // Ее сотрудники (30 максимум)
    Employee director;    // Директор
    CompanySize size;      // Размер компании
    unsigned int PeopleNumber; // количество сотрудников
    bool isBankrupt;       // Является ли компания банкротом (true / false)
};

int main() {
    Company comp[3]; // массив из 3х компаний

    comp[0].director = { 334567, 60, 150'000.0 };
    comp[0].isBankrupt = true;
    comp[0].PeopleNumber = 1;
    comp[0].size = CS_SMALL;

    comp[1] = comp[0]; // копируем информацию о первой компании во вторую
    comp[1].isBankrupt = false;
```

Какой размер занимает пустая структура?

Какие числа выведутся на экран?
Иногда спрашивают на собеседованиях по C++:
какой размер у пустого класса или структуры?

```
#include <iostream>

using namespace std;

struct TEmptyStruct
{

};

int main(int argc, char* argv[])
{
    TEmptyStruct s1;

    cout << sizeof(s1) << " " << sizeof(TEmptyStruct) << endl;
    return 0;
}
```



Объединения - union

Объединение – это группирование переменных, которые разделяют **одну и ту же область памяти**. В зависимости от интерпретации осуществляется обращение к той или другой переменной объединения. Все переменные, что включены в объединение начинаются с одной границы.

```
union EmployeeInfo { // Новый тип данных Информация о сотруднике
    long id;           // ID сотрудника
    unsigned short age; // его возраст
    double salary;      // его зарплата
};
```

Мы можем хранить только что-то одно: или id или age или salary. Если мы записали salary а пытаемся считать age то не понятно что это будет за число.

```
int main() {
    EmployeeInfo info;
    info.age = 25;
    std::cout << info.salary << std::endl; // Выведет на экран: -9.25596e+61
    std::cout << info.age << std::endl;    // Выведет на экран: 25
    std::cout << info.id << std::endl;      // Выведет на экран: -859045863

    info.salary = 175'000.0;
    std::cout << info.salary << std::endl; // Выведет на экран: 175000
    std::cout << info.age << std::endl;    // Выведет на экран: 0
    std::cout << info.id << std::endl;      // Выведет на экран: 0
}
```

Какой размер выведет операция sizeof(EmployeeInfo)?

Динамическая типизация и тип Variant

С помощью union можно реализовать динамическую типизацию и такой интересный тип данных в C++ как Variant, std::variant, QVariant и пр.

- **Динамическая типизация** — приём, используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов. Примеры языков с динамической типизацией — Smalltalk, Python, Objective-C, Ruby, PHP, Perl, JavaScript, Лисп.
- **Тип variant** в C++ позволяет нам работать с помощью языка со статической типизацией, как будто в ней возможна динамическая типизация. То есть переменная типа variant может хранить в разные моменты значения разных типов.



Битовые поля структур

В языке C++ есть возможность задавать элементам структур **определённое количество памяти в битах**. Типом элемента (его называют битовым полем) такой структуры может быть целочисленное (unsigned или signed).

У signed старший бит отведен под знак.

Для даты в формате: 31.01.20 создадим структуру с битовыми полями:

```
struct MyDate {  
    unsigned short Day : 5;    // можно хранить 0..31  
    unsigned short Month : 4; // можно хранить 0..15  
    unsigned short Year : 7;   // можно хранить 0..127  
};  
  
int main()  
{  
    MyDate date1 = { 31, 01, 20 };  
    std::cout << sizeof(date1);    // выведет на экран 2
```

Если бы мы хранили все 3 переменные просто в unsigned short то мы бы потратили 6 байт на одну такую дату. Но битовые поля позволяют сэкономить память.

Битовые поля структур — битовые флаги

Если отдать полю структуры только **один бит**, то у нас получится **битовый флаг**, со значениями (0 или 1):

```
struct MyBoolean
{
    unsigned int Flag : 1; // битовый флаг
};

int main()
{
    MyBoolean mybool;

    mybool.Flag = 1; // можно записать только 0 или 1 так как мы дали полю только 1 бит.

    return 0;
}
```

Последнее задание в **ДЗ** будет с битовыми полями (флагами).

Целочисленные типы из stdint.h

C type	stdint.h type	Bits	Sign	Range
unsigned char	uint8_t	8	Unsigned	0 .. 255
char	int8_t	8	Signed	-128 .. 127
unsigned short	uint16_t	16	Unsigned	0 .. 65,535
short	int16_t	16	Signed	-32,768 .. 32,767
unsigned int	uint32_t	32	Unsigned	0 .. 4,294,967,295
int	int32_t	32	Signed	-2,147,483,648 .. 2,147,483,647
unsigned long long	uint64_t	64	Unsigned	0 .. 18,446,744,073,709,551,615
long long	int64_t	64	Signed	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807

```
#include <stdint.h>
```

```
int main()
```

```
{
```

```
    uint8_t var0 = 255;           // аналог unsigned char
```

```
    uint16_t var1 = 100;          // аналог unsigned short int
```

```
    int64_t var2 = 1'000'000'000; // аналог long long
```

```
    return 0;
```

```
}
```

Венгерская нотация в программировании

Венгёрская нотация — соглашение об именовании переменных, констант и прочих идентификаторов в коде программ. Своё название венгерская нотация получила благодаря программисту компании Microsoft венгерского происхождения Чарльзу Симони.

Суть венгерской нотации сводится к тому, что **имена идентификаторов предваряются** заранее оговорёнными **префиксами**, состоящими из одного или нескольких символов. При этом, как правило, ни само наличие префиксов, ни их написание не являются требованием языков программирования, и у каждого программиста (или коллектива программистов) они могут быть своими.

```
struct TPerson {  
    unsigned int nAge;  
    char aName[30];  
    bool bWorking;  
};
```

Префикс	Сокращение от	Смысл	Пример
s	string	строка	sClientName
sz	zero-terminated string	строка, ограниченная нулевым символом	szClientName
n, i	int	целочисленная переменная	nSize, iSize
l	long	длинное целое	lAmount
b	boolean	булева переменная	bIsEmpty
a	array	массив	aDimensions
p	pointer	указатель	pBox
lp	long pointer	двойной (дальний) указатель	lpBox
r	reference	ссылка	rBoxes
h	handle	дескриптор	hWindow
m_	member	переменная-член	m_sAddress
g_	global	глобальная переменная	g_nSpeed
C	class	класс	CString
T	type	тип	TObject
I	interface	интерфейс	IDispatch

Вебинар 2. Домашнее задание.

В одном main.cpp файле / проекте:

1. Создать и инициализировать переменные пройденных типов данных (short int, int, long long, char, bool, float, double).
2. Создать **перечисление** (enum) с возможными вариантами символов для игры в крестики-нолики.
3. Создать **массив**, способный содержать значения такого перечисления и инициализировать его.
4. * Создать **структуру** (struct) данных «Поле для игры в крестики-нолики» и снабдить его всеми необходимыми свойствами (подумайте что может понадобиться).
5. ** Создать **структуру** (struct MyVariant) объединяющую: union MyData (int, float, char) и 3-и битовых поля (флага) указывающими какого типа значение в данный момент содержится в объединении (isInt, isFloat, isChar).
Продемонстрировать пример использования в коде этой структуры.

Для программирования используйте установленную среду программирования (IDE). Если задания 4 и 5 кажутся сложными постарайтесь сделать первые 3.

Основы C++. Вебинар №2.

Успеха с домашним заданием!



GeekBrains