

Структура языка программирования. Система типов

Определения:

Тип данных – множество значений и операций над этими значениями.
(IEEE Std 1320.2-1998).

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;
- множество значений, которые могут принимать величины этого типа;
- операции и функции, которые можно применять к величинам этого типа.

Типы данных:

Неинтерпретируемые:	бит; байт; кубит; слово
Числовые:	целый; с фиксированной запятой; с плавающей запятой; рациональный; комплексный; длинный;
Текстовые:	символьный; строковый
Ссылочные:	адрес; ссылка; указатель; обертка (объектный тип данных для хранения значения неobjектного типа. например, в java класс <code>integer</code> является надстройкой для примитивного типа <code>int</code>)
Абстрактные:	массив; список; очередь; стек; ассоциативный массив (словарь); множество; граф
Композитные:	класс; тип-произведение (запись кортеж структура); объект; объединение; упорядоченная пара

Бит (bit; от англ. binary digit – двоичное число; также игра слов: англ. bit – кусочек, частица) – единица измерения количества информации.

Один бит информации – это символ или сигнал, который может принимать два значения: включено или выключено; да или нет; высокий или низкий; заряженный или незаряженный; в двоичной системе исчисления это 1 (единица) или 0 (ноль).

Байт (byte) – единица хранения и обработки цифровой информации; совокупность битов, обрабатываемая компьютером одновременно.

В современных вычислительных системах байт состоит из 8 бит и, соответственно, может принимать одно из 256 (от 0 до 255) различных значений (состояний, кодов). В большинстве вычислительных архитектур **байт** – это минимальный независимо адресуемый набор данных.

Кубит (q-бит, кьюбит, кубит; от англ. quantum bit) – наименьшая единица информации в квантовом компьютере (аналог бита в обычном компьютере), используемая для квантовых вычислений.

Машинное слово – это фрагмент данных фиксированного размера, обрабатываемый как единое целое с помощью набора команд или аппаратного обеспечения процессора.

Количество бит в машинном слове – размер слова (он же ширина или длина слова) – является важной характеристикой любой конкретной архитектуры процессора или компьютерной архитектуры.

1. Типизация в языках программирования (как различные языки распознают типы переменных)

Система типов – совокупность правил в языках программирования, назначающих свойства, называемыми типами, различным конструкциям, составляющим программу (переменным, выражениям, функциям, модулям) для возможности выполнения проверки типов во время компиляции или во время выполнения,

Основные функции системы типов данных:

- **обеспечение безопасности:** проверяется каждая операция на получение аргументов именно тех типов, для которых она имеет предназначена;
- **оптимизация:** на основе типа выбирается способ эффективного хранения значения и алгоритмов его обработки;
- **документация:** подчеркивается намерения программиста;
- **абстракция:** использование типов данных высокого уровня позволяет программисту думать о значениях как о высокоуровневым сущностям, а не как о наборе битов.

а. Контроль типов и системы типизации

Два основных вида типизации в языках программирования:

нетипизированные (языки ассемблера, Forth и Brainfuck (эзотерический язык), другие)
(**бестиповые**)

типизированные (C, C++, Java, Python, Scala, Rust, PHP, C#, F#, другие)

В **бестиповых языках программирования** все сущности — это **последовательности битов**, различной длины.

Примеры:

- в ассемблере единственный тип — это последовательность битов;
- в лямбда-исчислении единственный тип — это функция.

Выравнивание данных в оперативной памяти компьютеров:

требование для объектов определенного типа располагаться на границах ячеек памяти с адресами, кратными своему же размеру.

По умолчанию компилятор выравнивает элементы структуры, класса по значению размера типа:

- bool и char в однобайтовых границах;
- short на 2-байтовых границах;
- int, long и float в пределах 4-байтового диапазона;
- long long, double и long double в пределах 8-байтовых границ.

Типизированные языки разделяются на несколько *пересекающихся* категорий:

статическая (static) / динамическая (dynamic) типизация	
<p>Статическая: типы переменных и функций устанавливаются на этапе компиляции. Типы данных ассоциируются с переменными, а не с конкретными значениями. <i>Примеры:</i> C, C++, Java, C#</p>	<p>Динамическая: типы проверяются динамически (во время исполнения программы). Типы данных ассоциируются с конкретными значениями. <i>Примеры:</i> Python, JavaScript, Ruby</p>
строгая (strongly) / нестрогая (weakly typed) типизация	
<p>Строгая: языки не позволяют смешивать в одном выражении различные типы; не выполняют автоматические неявные преобразования; запрещены изменения типа переменной в течение времени ее жизни. <i>Примеры:</i> Java, Python, Haskell, Lisp</p>	<p>Нестрогая: языки выполняют множество неявных преобразований автоматически, даже если может произойти потеря точности или преобразование неоднозначно. <i>Примеры:</i> JavaScript, Visual Basic, PHP</p>
явная / неявная типизация	
<p>Явная: тип новых переменных / функций / их аргументов нужно задавать явно (частный случай статической типизации). <i>Примеры:</i> C, C++, Objective-C, C#</p>	<p>неявная: определение типа выполняет компилятор / интерпретатор. <i>Примеры:</i> PHP, Lua, JavaScript</p>

В C/C++ размер переменной любого типа данных зависит от компилятора и/или архитектуры компьютера. Фактический размер переменных может отличаться на разных компьютерах. Для его определения используют оператор `sizeof`.

Стандарт задает следующее *отношение* размера между целыми типами:

```
1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)
<= sizeof(long long)
```

и для переменных с плавающей запятой:

```
sizeof(float) <= sizeof(double) <= sizeof(long double)
```

Также поддерживаются целочисленные типы с указанием их размера:

```
__int8, __int16, __int32, __int64 и __int8, __int16, __int32,
__int64.
```

2. Вывод типов

В стандарте C++11 введено новое ключевое слово **auto** для определения *явно* инициализируемой переменной.

Создается переменная, тип которой выводится из *инициализирующего значения*:

```
auto variable1 = 5;
auto variable2 = 2.5;
```

3. Преобразование типов:

- автоматическое (неявное) преобразование;
- явное преобразование.

<i>явное</i>	задается программистом в тексте программы с помощью: <ul style="list-style-type: none">– конструкции языка;– функции, принимающей значение одного типа и возвращающей значение другого типа.
<i>неявное</i>	выполняется автоматически транслятором (компилятором или интерпретатором) по правилам, описанным в стандарте языка.

Стандарты большинства языков запрещают *неявные* преобразования.

4. Автоматическое (неявное) преобразование типов:

Для базовых типов

bool, [unsigned/signed] char, short, int, long, float, double, long double

преобразование типов выполняется без потери точности.

Пример. Безопасное преобразование:

символ 'а' → целое 0x61 → символ 'а'

Если выбранное преобразование является *расширяющим*, компилятор выполняет его, не информируя о выполнении такого преобразования. Расширяющие преобразования всегда являются надежными.

Если преобразование является *сужающим*, компилятор выдает предупреждение о возможной потере данных.

Происходит ли фактическая потеря данных, зависит от *фактических* значений. Рекомендуется рассматривать это предупреждение как ошибку.

! Если компилятору не удастся найти допустимое преобразование, то выдается ошибка и объектный код не создается.

а. расширяющее преобразование.

```

4  #include "stdafx.h"
5
6  int main()
7  {
8      bool b1 = false;
9      bool b2 = true;
10     char c1 = b1;
11     char c2 = b2;
12     short s1 = c1;
13     short s2 = c2;
14     int i1 = s1;
15     int i2 = s2;
16     float f1 = i1;
17     float f2 = i2;
18     double d1 = f1;
19     double d2 = f2;
20     int i = 3.14;    // i == 3
21     //int i{ 3.14 }; // Ошибка компиляции
22     char c = 128;   // c == 0x80
23
24     return 0;
25 }

```

Память 1

Адрес: 0x0115F91F ← &c

0x0115F91F	80	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	03	БМММММММ.
0x0115F929	00	00	00	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	...ММММММ

Контрольные значения 1

Поиск (Ctrl+E) 🔍 ← → Глубина поиска

Имя	Значение	Тип
b1	false	bool
b2	true	bool
c1	0x00 '\0'	char
c2	0x01 '\x1'	char
s1	0x0000	short
s2	0x0001	short
i1	0x00000000	int
i2	0x00000001	int
f1	0.00000000	float
f2	1.00000000	float
d1	0.0000000000000000	double
d2	1.0000000000000000	double
i	0x00000003	int
c	0x80 'Б'	char
&c	0x0115f91f "БМММММММ..."	char *
'Б'	0x80 'Б'	char

Список ошибок

Все решение 0 Ошибки 4 Предупреждения 0 Сообщения Сборка и IntelliSense

Код	Описание	Проект	Файл	Ст...
C4244	инициализация: преобразование "int" в "float", возможна потеря данных	Преобразование типов	Преобразование типов....	16
C4244	инициализация: преобразование "int" в "float", возможна потеря данных	Преобразование типов	Преобразование типов....	17
C4244	инициализация: преобразование "double" в "int", возможна потеря данных	Преобразование типов	Преобразование типов....	20
C4309	инициализация: усечение константного значения	Преобразование типов	Преобразование типов....	22

Параметр «Уровень предупреждений»:

Конфигурация: Активная (Debug) Платформа: Активная (Win32) Диспетчер конфигураций...

Общие свойства

Свойства конфигурации

Общие

Отладка

Каталоги VC++

C/C++

Общие

Оптимизация

Препроцессор

Создание кода

Язык

Предварительно отко

Выходные файлы

Информация об исхо

Дополнительно

Все параметры

Командная строка

Компоновщик

Инструмент манифеста

Генератор XML-документ

Информация об исходно

События построения

Настраиваемый этап пос

Дополнительные каталоги включаемых ф

Дополнительные каталоги #using

Формат отладочной информации База данных программы для операции "Изменить и продолжи

Поддержка общезыковой среды выполн

Использовать расширение среды выполн

Отключить загрузочный баннер Да (/nologo)

Уровень предупреждений **Уровень3 (/W3)**

Обрабатывать предупреждения Отключить все предупреждения (/W0)

Проверки SDL Уровень1 (/W1)

Многопроцессорная ком Уровень2 (/W2)

Уровень3 (/W3)

Уровень4 (/W4)

Включить все предупреждения (/Wall)

<наследовать от родителя или от значений по умолчанию для проекта>

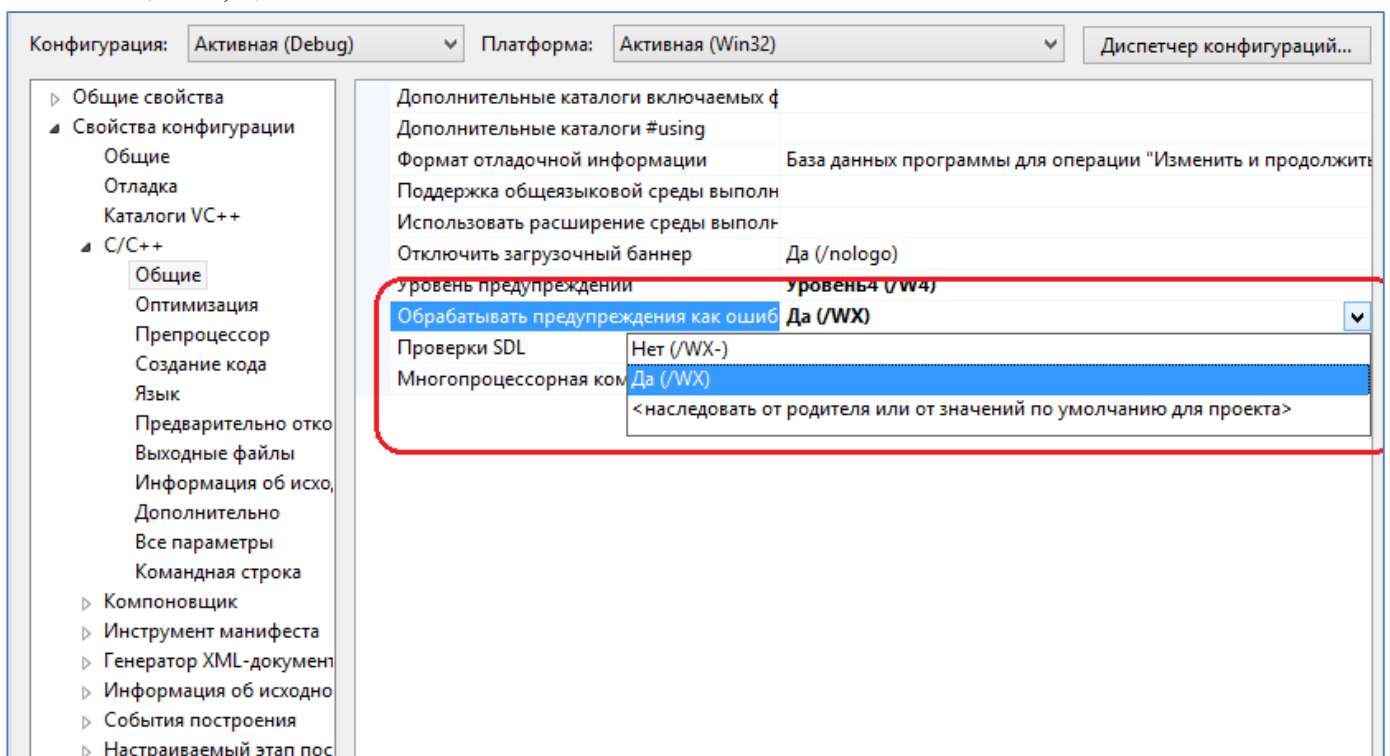
Список ошибок						
Все решение		0 Ошибки	8 Предупреждения	0 Сообщения	Сборка и IntelliSense	
	Код	Описание	Проект	Файл	Ст...	
!	C4244	инициализация: преобразование "int" в "float", возможна потеря данных	Преобразование типов	Преобразован...	16	
!	C4244	инициализация: преобразование "int" в "float", возможна потеря данных	Преобразование типов	Преобразован...	17	
!	C4244	инициализация: преобразование "double" в "int", возможна потеря данных	Преобразование типов	Преобразован...	20	
!	C4309	инициализация: усечение константного значения	Преобразование типов	Преобразован...	22	
!	C4189	d1: локальная переменная инициализирована, но не использована	Преобразование типов	Преобразован...	18	
!	C4189	c: локальная переменная инициализирована, но не использована	Преобразование типов	Преобразован...	22	
!	C4189	i: локальная переменная инициализирована, но не использована	Преобразование типов	Преобразован...	20	
!	C4189	d2: локальная переменная инициализирована, но не использована	Преобразование типов	Преобразован...	19	

Пример:

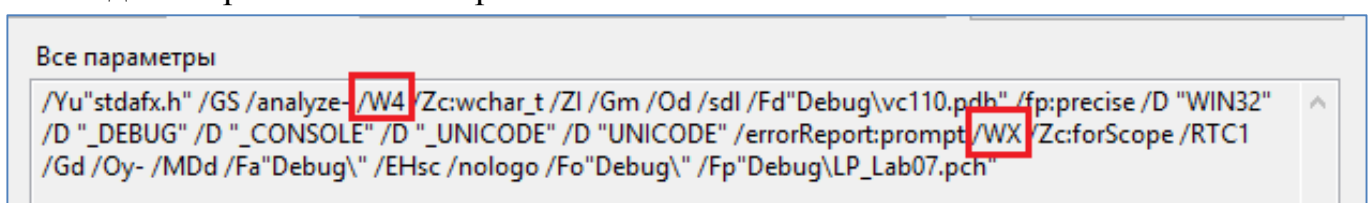
```
unsigned int u3 = 0 - 1;
cout << u3 << endl;           // вывод: 4294967295
```

! Компилятор *не предупреждает* о неявных преобразованиях между целыми типами со знаком и без знака.

Если установлен параметр компиляции «Обрабатывать предупреждения как ошибки (/WX)», то объектный код не создается:



Командная строка компилятора в MSVS:



Окно вывода:

	Описание	Файл	Строка
✖ 1	error C2220: предупреждение обработано как ошибка - файл "object" не создан	преобразование типов.cpp	17
⚠ 2	warning C4244: инициализация: преобразование "int" в "float", возможна потеря данных	преобразование типов.cpp	17
⚠ 3	warning C4244: инициализация: преобразование "int" в "float", возможна потеря данных	преобразование типов.cpp	18
⚠ 4	warning C4244: инициализация: преобразование "double" в "int", возможна потеря данных	преобразование типов.cpp	21
⚠ 5	warning C4309: инициализация: усечение константного значения	преобразование типов.cpp	23
⚠ 6	warning C4100: argv: неиспользованный формальный параметр	преобразование типов.cpp	6
⚠ 7	warning C4100: argc: неиспользованный формальный параметр	преобразование типов.cpp	6
⚠ 8	warning C4189: i: локальная переменная инициализирована, но не использована	преобразование типов.cpp	21
⚠ 9	warning C4189: d2: локальная переменная инициализирована, но не использована	преобразование типов.cpp	20
⚠ 10	warning C4189: c: локальная переменная инициализирована, но не использована	преобразование типов.cpp	23
⚠ 11	warning C4189: d1: локальная переменная инициализирована, но не использована	преобразование типов.cpp	19

б. преобразования указателей

Указатель `ps` увеличивается на три элемента, теперь он указывает на 4-ый элемент массива со значением "p".

Пример:

```
char s[]{"Help"};
char* ps = s + 3;
cout << ps << endl;           // вывод 4-го элемента: p
```

Имя массива в стиле C неявно преобразуется в указатель на первый элемент массива.

Пример:

```
char s[]{"Help"};
cout << s + 3 << endl;        // вывод 4-го элемента: p
```

с. сужающее преобразование

```
double d1 = 321.12345678912345678;
float f1 = d1;
int i1 = f1;
short s1 = i1;
char c1 = s1;
bool b1 = c1;
```

Контрольные значения 1		
Имя	Значение	Тип
d1	321.12345678912345	double
f1	321.123444	float
i1	321	int
s1	321	short
c1	65 'A'	char
b1	true	bool

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
    signed    int i1 = -100000;
    unsigned  int i2 = i1;

    return 0;
}
```

Имя	Значение	Тип
i1	-100000	int
i2	4294867296	unsigned int

```
std::cout << std::endl<< std::endl;
int i1 = 0, i2 = 100, i3 = -100;
if (i1) std::cout << "int i1 =" << i1 <<"-->true";
else    std::cout << "int i1 =" << i1 <<"-->false";
std::cout << std::endl;

if (i2) std::cout << "int i2 =" << i2 <<"-->true";
else    std::cout << "int i2 =" << i2 <<"-->false";
std::cout << std::endl;

if (i3) std::cout << "int i3 =" << i3 <<"-->true";
else    std::cout << "int i3 =" << i3 <<"-->false";
std::cout << std::endl<< std::endl;

float f1 = 0, f2 = 123.321, f3 = -123.321;
if (f1) std::cout << "float f1 =" << f1 <<"-->true";
else    std::cout << "float f1 =" << f1 <<"-->false";
std::cout << std::endl;

if (f2) std::cout << "float f2 =" << f2 <<"-->true";
else    std::cout << "float f2 =" << f2 <<"-->false";
std::cout << std::endl;

if (f3) std::cout << "float f3 =" << f3 <<"-->true";
else    std::cout << "float f3 =" << f3 <<"-->false";
std::cout << std::endl<<std::endl;

char c1 = 0x00, c2 = 'f';
if (c1) std::cout << "char c1 =" << c1 <<"-->true";
else    std::cout << "char c1 =" << c1 <<"-->false";
std::cout << std::endl;

if (c2) std::cout << "char c2 =" << c2 <<"-->true";
else    std::cout << "char c2 =" << c2 <<"-->false";
std::cout << std::endl<<std::endl;

return 0;
```

```
c:\users\user p...
int i1 = 0-->false
int i2 =100-->true
int i3 =-100-->true

float f1 = 0-->false
float f2 =123.321-->true
float f3 =-123.321-->true

char c1 = -->false
char c2 =f-->true
```

5. Явное преобразование:

часто применяется для указания того, что преобразование делается осознано.

Позволяет *отключить* механизм неявных преобразований типов посредством явного указания в исходном тексте программы требуемого преобразования.

```
#include "stdafx.h"
#include <iostream>
int _tmain(int argc, _TCHAR* argv[])
{
    float fx = 3.143334, fy = 223.123, fc;

    fc = fx - (int)fx;    // дробная часть
    fx = (int)fx;         // округление до нижнего целого
    fy = (int)(fy+1);     // округление до верхнего целого

    return 0;
}
```

fc	0.143333912	float
fx	3.000000000	float
fy	224.0000000	float

а. явное преобразование:

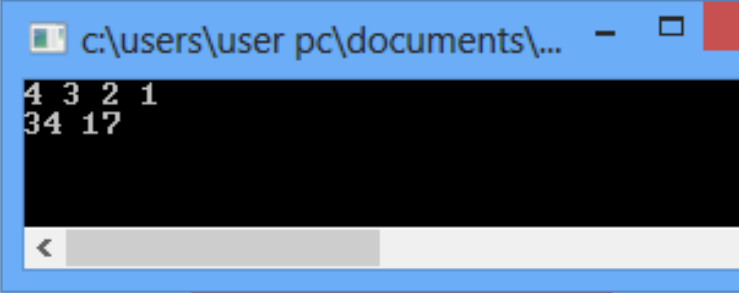
чаще всего применяется для приведения **void*** к некоторому типу.

```
#include <iomanip>

int _tmain(int argc, _TCHAR* argv[])
{
    struct MEM { char mem[4]; };
    void *v1, *v2;
    int x = 0x01020304;
    short y = 0x1122;
    v1 = &x;
    v2 = &y;

    for (int k = 0; k < 4; k++) std::cout<< (int)((MEM*)v1)->mem[k] << " ";
    std::cout<<std::endl;

    for (int k = 0; k < 2; k++) std::cout<< (int)((MEM*)v2)->mem[k] << " ";
    return 0;
}
```



в. Явное преобразование приведения типов в C++:

приведение типов **const_cast**;

приведения типов на этапе выполнения программы **dynamic_cast**;

приведения несовместимых типов **reinterpret_cast**;

приведения типов на этапе компиляции программы **static_cast** (может отслеживать недопустимые преобразования).

Для выполнения явных преобразований типов (explicit type conversion) применяется оператор **static_cast**

Синтаксис:

`static_cast<type>(value)`

Оператор преобразует значение в круглых скобках – **value** к типу, который указан в угловых скобках – **type**. Слово **static** в названии оператора указывает, что приведение проверяется статически, то есть во время компиляции.

Пример:

```
float fx = 3.14334;
int i;
i = static_cast<int>(fx);
std::cout << i << std::endl;           // вывод: 3
```

6. Константное выражение:

выражение, которое должно быть вычислено на этапе компиляции.

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>

int _tmain(int argc, _TCHAR* argv[])
{
    const int k = 5*25;

    int m = k/5;

    return 0;
}
```

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>

int _tmain(int argc, _TCHAR* argv[])
{
    const int k = 5*25;

    int m = k/5;

    k = 2*m;

    return 0;
}
```

❌ 1 error C3892: k: невозможно присваивать значения переменной, которая объявлена как константа

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>

int _tmain(int argc, _TCHAR* argv[])
{
    const int k = 5*25;

    int m = k/5;

    int* v = &k;

    return 0;
}
```

❌ 1 error C2440: инициализация: невозможно преобразовать "const int *" в "int *"

```

#include "stdafx.h"
#include <iostream>
#include <iomanip>

int _tmain(int argc, _TCHAR* argv[])
{
    const int k = 5*25;

    int m = k/5;

    const int* v = &k;

    return 0;
}

```

+	v	0x00bcfa68 {125}	const int *
---	---	------------------	-------------

```

#include "stdafx.h"
#include <iostream>
#include <iomanip>

int _tmain(int argc, _TCHAR* argv[])
{
    const int k = 5*25;

    int m = k/5;

    const int* v = &k;

```

```

    (*v) = 15;

```

1 error C3892: v: невозможно присваивать значения переменной, которая объявлена как константа

В C++11 введено новое ключевое слово **constexpr**, которое позволяет пользователю гарантировать, что функция или конструктор объекта возвращает константу времени компиляции.

Пример:

```
constexpr int GiveFive() {return 5;}  
int some_value[GiveFive() + 7];      // разрешено в C++11
```

Использование constexpr:

- a) **constexpr**-функция должна возвращать значение;
- b) тело функции должно быть вида `return <выражение>;`;
- c) выражение должно состоять из констант и/или вызовов других **constexpr**-функций;
- d) **constexpr**-функция не может использоваться до определения в текущей единице компиляции.

7. Инициализация переменных (памяти):

присвоение значения в момент объявления переменной;
как правило, применяется литералы.

Отличие от присвоения: при присвоении явно перемещаются данные.

Инициализация массивов, структур. Функциональный вид инициализации:

```
#include "stdafx.h"  
#include <iostream>  
#include <iomanip>  
// функциональная форма инициализации  
int _tmain(int argc, _TCHAR* argv[])  
{  
    int k(5*25);  
    int l(k*25);  
    float f(25.18f);  
    float *pf(&f);  
    char c('a');  
  
    return 0;  
}
```

8. Область видимости переменных в C++:

доступность переменных по их идентификатору в разных частях (блоках программы).

9. Область видимости переменных в C++:

- переменная должна быть объявлена до ее использования;
- переменная объявленная во внутреннем блоке (локальная переменная {...}) не доступна во внешнем;
- переменная объявленная во внешнем блоке доступна во внутреннем;
- во внутреннем блоке переменная может быть переобъявлена.

Область видимости переменной (идентификатора) зависит от места ее объявления в тексте программы.

Область действия идентификатора — это часть программы, в которой его можно использовать для доступа к связанной с ним области памяти.

В зависимости от области действия переменная может быть **локальной** или **глобальной**.

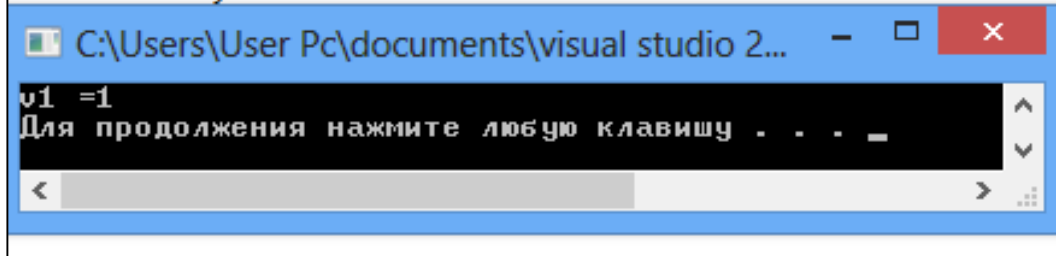
Глобальная переменная:

```
#include "stdafx.h"
#include <locale>
#include <iostream>

int v1 = 1;

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<<"v1 ="<< v1 <<std::endl;

    system("pause");
    return 0;
}
```



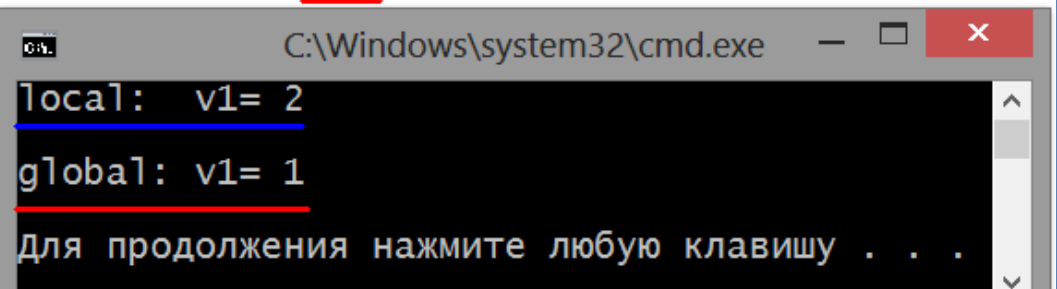

```
#include "stdafx.h"
#include <iostream>

int v1 = 1;

int _tmain(int argc, _TCHAR* argv[])
{
    int v1 = 2;
    std::cout << "local: v1= " << v1 << std::endl << std::endl;

    std::cout << "global: v1= " << ::v1 << std::endl << std::endl;

    return 0;
}
```



```
local: v1= 2
global: v1= 1
Для продолжения нажмите любую клавишу . . .
```

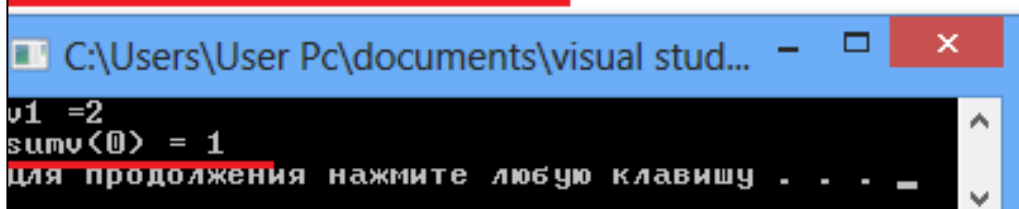
```
#include "stdafx.h"
#include <locale>
#include <iostream>

int v1 = 1;
int sumv(int k);

int _tmain(int argc, _TCHAR* argv[])
{
    int v1 = 2;
    std::cout<<"v1 ="<< v1 <<std::endl;
    std::cout<<"sumv(0) = "<<sumv(0)<<std::endl;

    system("pause");
    return 0;
}

int sumv(int k) {return v1+k;}
```



```
v1 =2
sumv(0) = 1
Для продолжения нажмите любую клавишу . . .
```


10. Пространство имен:

именованная область видимости (применяется для разрешения конфликтов имен).

Пример области видимости для языков разметки:

В HTML областью видимости *имени элемента управления* является форма от тега `<form>` до тега `</form>`

11. Пространство имен в C++:

`namespace`, `using`, псевдонимы пространства имен.

Сопоставление имени с его объявлением называется **разрешением**.

Области видимости и разрешение имен – понятия **времени компиляции**.

Ключевое слово `namespace` позволяет разделить глобальное пространство имен путем создания некоторой объявленной (декларативной) области в нем.

Для доступа к определенным внутри области объектам, используется

✓ **оператор разрешения видимости «::»**.

С помощью ключевого слова `using` расширяются области видимости всех членов пространства имен.

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

namespace A
{
    int x = 1;
    char c = 'A';
    float func_mul(float x, float y){return x*y;};
};

float func_mul(float x, float y){return x*y;};

int _tmain(int argc, _TCHAR* argv[])
{
    int x = 1;
    char c = 'A';

    std::cout<< " c = "<<c<<" x = " << x << " func_mul(3.0, 3.3) = " <<func_mul(3.0, 3.3)<< std::endl;

    std::cout<< " A::c = "<<A::c<<" A::x = " << A::x << " A::func_mul(4.0, 4.4) = " << A::func_mul(4.0, 4.4) <<std::endl;

    system("pause");

    return 0;
}
```

Конфликт имен:

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

namespace A
{
    int x = 1;
    char c = 'A';
    float func_mul(float x, float y){return x*y;};
};

float func_mul(float x, float y){return x*y;};

using namespace A;

int _tmain(int argc, _TCHAR* argv[])
{
    int x = 1;
    char c = 'A';

    std::cout<< " c = "<<c<<" x = " << x << " func_mul(3.0, 3.3) = " <<func_mul(3.0, 3.3)<< std::endl;

    std::cout<< " A::c = "<<A::c<<" A::x = " << A::x << " A::func_mul(4.0, 4.4) = " << A::func_mul(4.0, 4.4) <<std::endl;

    system("pause");

    return 0;
}
```

✖ 1 error C2668: func_mul: неоднозначный вызов перегруженной функции

ℹ 3 IntelliSense: существует более одного экземпляра перегруженной функции "func_mul", соответствующего списку аргументов:
функция "func_mul(float x, float y)"
функция "A::func_mul(float x, float y)"
типы аргументов: (double, double)

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

namespace A
{
    int x = 1;
    char c = 'A';
    float func_mul(float x, float y){return x*y;};
};

//float func_mul(float x, float y){return x*y;};

using namespace A;

int _tmain(int argc, _TCHAR* argv[])
{
    int x = 1;
    char c = 'A';

    //std::cout<< " c = "<<c<<" x = " << x << " func_mul(3.0, 3.3) = " <<func_mul(3.0, 3.3)<< std::endl;

    std::cout<< " A::c = "<<A::c<<" A::x = " << A::x << " A::func_mul(4.0, 4.4) = " << A::func_mul(4.0, 4.4) <<std::endl;

    system("pause");

    return 0;
}
```

Разрешение конфликта: задание области видимости В

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

namespace A
{
    int x = 1;
    char c = 'A';
    float func_mul(float x, float y){return x*y;};
};

namespace B
{
    int x = 1;
    char c = 'B';
    float func_mul(float x, float y){return x*y;};
};

int _tmain(int argc, _TCHAR* argv[])
{
    int x = 1;
    char c = 'A';

    std::cout<< "B::c = "<<B::c<<" B::x = "<<B::x<<" B::func_mul(3.0, 3.3) = " <<B::func_mul(3.0, 3.3)<< std::endl;

    std::cout<< "A::c = "<<A::c<<" A::x = "<< A::x<<" A::func_mul(4.0, 4.4) = " << A::func_mul(4.0, 4.4) <<std::endl;

    system("pause");

    return 0;
}
```

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

namespace A
{
    int x = 1;
    char c = 'A';
    float func_mulA(float x, float y){return x*y;};
};

namespace B
{
    int y = 1;
    char b = 'B';
    float func_mulB(float x, float y){return x*y;};
};

using namespace A;
using namespace B;

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<< "B::b = "<<b<<" B::y = "<<y<<" B::func_mulB(3.0, 3.3) = " <<func_mulB(3.0, 3.3)<< std::endl;

    std::cout<< "A::c = "<<c<<" A::x = "<<x<<" A::func_mul(4.0, 4.4) = " << func_mulA(4.0, 4.4) <<std::endl;

    system("pause");

    return 0;
}
```

```

#include "stdafx.h"
#include <iostream>

namespace A
{
    int x = 1;
    char c = 'A';
};

namespace B
{
    float func_mulB(float x, float y){return x*y;};
};

namespace A
{
    float func_mulA(float x, float y){return x*y;};
};

namespace B
{
    int y = 1;
    char b = 'B';
};

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<< "B::b = "<<B::b<<" B::y = "<<B::y<<" B::func_mulB(3.0, 3.3) = " <<B::func_mulB(3.0, 3.3)<< std::endl;
    std::cout<< "A::c = "<<A::c<<" A::x = "<<A::x<<" A::func_mulA(4.0, 4.4) = " << A::func_mulA(4.0, 4.4) <<std::endl;
    system("pause");
    return 0;
}

```

Различные области видимости следует размещать в отдельных сpp-файлах

```

#include "stdafx.h"
#include <iostream>

namespace A {int x = 1; char c = 'A';float func_mulA(float x, float y);}
namespace B {int y = 1; char b = 'B'; float func_mulB(float x, float y);}

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<< "B::b = "<<B::b<<" B::y = "<<B::y<<" B::func_mulB(3.0, 3.3) = "
    std::cout<< "A::c = "<<A::c<<" A::x = "<<A::x<<" A::func_mulA(4.0, 4.4) = "
    system("pause");
    return 0;
}

```

LPLab06

- Внешние зависимости
- Заголовочные файлы
 - stdafx.h
 - targetver.h
- Файлы исходного кода
 - LPLab06.cpp
 - NS_A.cpp
 - NS_B.cpp
 - stdafx.cpp
- Файлы ресурсов
 - ReadMe.txt

```

#include "stdafx.h"

namespace A
{
    float func_mulA(float x, float y){return x*y;};
};

```

```

#include "stdafx.h"

namespace B
{
    float func_mulB(float x, float y){return x*y;};
};

```

Задание псевдонимов для областей видимости

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

namespace A {int x = 1; char c = 'A'; float func_mulA(float x, float y);}
namespace B {int y = 1; char b = 'B'; float func_mulB(float x, float y);}

namespace BB = B; //псевдоним
namespace AA = A; //псевдоним
namespace BB1 = B; //псевдоним

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<< "B::b = "<<B::b<<" B::y = "<<B::y<<" B::func_mulB(3.0, 3.3) = " <<B::func_mulB(3.0, 3.3)<< std::endl;
    std::cout<< "BB::b = "<<BB::b<<"BB::y = "<<BB::y<<" BB::func_mulB(3.0, 3.3) = " <<BB::func_mulB(3.0, 3.3)<< std::endl;
    std::cout<< "BB1::b = "<<BB1::b<<"BB1::y = "<<BB1::y<<" BB1::func_mulB(3.0, 3.3) = " <<BB1::func_mulB(3.0, 3.3)<< std::endl;
    std::cout<< "A::c = "<<A::c<<" A::x = "<<A::x<<" A::func_mulA(4.0, 4.4) = " << A::func_mulA(4.0, 4.4) <<std::endl;
    std::cout<< "AA::c = "<<AA::c<<" AA::x = "<<AA::x<<" AA::func_mulA(4.0, 4.4) = " << AA::func_mulA(4.0, 4.4) <<std::endl;
    system("pause");
    return 0;
}
```

Вложенные области видимости

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

namespace A
{
    int x = 1; char c = 'A';
    float func_mulA(float x, float y){return x*y;};
    namespace B
    {
        int y = 1;
        char b = 'B';
        float func_mulB(float x, float y){return x*y;};
    };
};

namespace B
{
    int y = 1;
    char b = 'B';
    float func_mulB(float x, float y){return x*y;};
};

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<< "B::b = "<<B::b<<" B::y = "<<B::y<<" B::func_mulB(3.0, 3.3) = " <<B::func_mulB(3.0, 3.3)<< std::endl;
    std::cout<< "A::c = "<<A::c<<" A::x = "<<A::x<<" A::func_mulA(4.0, 4.4) = " << A::func_mulA(4.0, 4.4) <<std::endl;
    std::cout<< "A::B::b = "<<A::B::b<<" A::B::y = "<<A::B::y<<" A::B::func_mulB(3.0, 3.3) = " <<A::B::func_mulB(3.0, 3.3)<< std::endl;
    system("pause");
    return 0;
}
```

Безымянная область видимости

```
#include <iostream>

namespace A
{
    int x = 1; char c = 'A';
    float func_mulA(float x, float y){return x*y;};
};

namespace
{
    int x = 1;
    char c = 'X';
    float func_mulA(float x, float y){return x*y;};
};

namespace B
{
    int y = 1;
    char b = 'B';
    float func_mulB(float x, float y){return x*y;};
};

namespace
{
    int y = 1;
    char b = 'Y';
    float func_mulB(float x, float y){return x*y;};
};

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<< "B::b = "<<B::b<<" B::y = "<<B::y<<" B::func_mulB(3.0, 3.3) = " <<B::func_mulB(3.0, 3.3)<< std::endl;
    std::cout<< "A::c = "<<A::c<<" A::x = "<<A::x<<" A::func_mulA(4.0, 4.4) = " << A::func_mulA(4.0, 4.4) <<std::endl;
    std::cout<< "::c = "<<::c<<" ::x = "<<::x<<" ::func_mulA(5.0, 5.5) = " <<::func_mulA(5.0, 5.5) <<std::endl;
    std::cout<< "::b = "<<::b<<" ::y = "<<::y<<" ::func_mulB(5.0, 5.5) = " <<::func_mulB(5.0, 5.5) <<std::endl;
    system( "pause" );
    return 0;
}
```