

Структура языка программирования.**1. 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;
```

2. Преобразование типов:

автоматическое преобразование, явное преобразование.

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

Стандарты большинства языков запрещают неявные преобразования.

3. 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 безопасного преобразования:

```
символ 'a' -> целое 0x41 -> символ 'a'
```

```

#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    bool b1 = false;
    bool b2 = true;
    char c1 = b1;
    char c2 = b2;
    short s1 = c1;
    short s2 = c2;
    int i1 = s1;
    int i2 = s2;
    float f1 = i1;
    float f2 = i2;
    double d1 = f1;
    double d2 = f2;
    int i = 3.14;    // i == 3
    //int i{ 3.14 }; // Ошибка компиляции
    char c = 128;    // c == -128

    return 0;
}

```

Контрольные значения 1

Имя	Значение	Тип
b2	true	bool
c1	0x00 '\0'	char
c2	0x01 '\x1'	char
s1	0x0000	short
s2	0x0001	short
i1	0x00000000	int
i2	0x00000001	int
f1	0.000000000	float
f2	1.000000000	float
d1	0.000000000000000000	double
d2	1.000000000000000000	double
i	0x00000003	int
c	0x80 'Б'	char

Локальные

Контрольные зн...

Результаты поис...

1	warning C4244: инициализация: преобразование "int" в "float", возможна потеря данных	преобразование типов.cp	17	1
2	warning C4244: инициализация: преобразование "int" в "float", возможна потеря данных	преобразование типов.cp	18	1
3	warning C4244: инициализация: преобразование "double" в "int", возможна потеря данных	преобразование типов.cp	21	1
4	warning C4309: инициализация: усечение константного значения	преобразование типов.cp	23	1

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 предупреж

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>

	Описание
! 1	warning C4244: инициализация: преобразование "int" в "float", возможна потеря данных
! 2	warning C4244: инициализация: преобразование "int" в "float", возможна потеря данных
! 5	warning C4189: d2: локальная переменная инициализирована, но не использована
! 6	warning C4189: d1: локальная переменная инициализирована, но не использована
! 3	warning C4100: argv: неиспользованный формальный параметр
! 4	warning C4100: argc: неиспользованный формальный параметр

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

Общие свойства

Свойства конфигурации

- Общие
- Отладка
- Каталоги VC++
- С/С++
 - 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 продолжить"

Поддержка общезыковой среды выполн

Использовать расширение среды выполн

Отключить загрузочный баннер

Да (/nologo)

уровень предупреждений

уровень4 (/W4)

Обрабатывать предупреждения как ошибки

Да (/WX)

Проверки SDL

Нет (/WX-)

Многопроцессорная ком

Да (/WX)

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

Все параметры

```

/Yu"stdafx.h" /GS /analyze- /W4 /Zc:wchar_t /ZI /Gm /Od /sdl /Fd"Debug\vc110.pdb" /fp:precise /D "WIN32"
/D "_DEBUG" /D "_CONSOLE" /D "_UNICODE" /D "UNICODE" /errorReport:prompt /WX /Zc:forScope /RTC1
/Gd /Oy- /MDd /Fa"Debug\" /EHsc /nologo /Fo"Debug\" /Fp"Debug\LP_Lab07.pch"

```

	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

```
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
```

4. Явное преобразование:

часто применяется для указания того, что преобразование делается осознано. Таким образом, механизм неявных преобразований может быть отключён посредством явного указания в тексте программы требуемого преобразования типов.

```
#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	3.00000000	float
fy	224.000000	float

5. Явное преобразование:

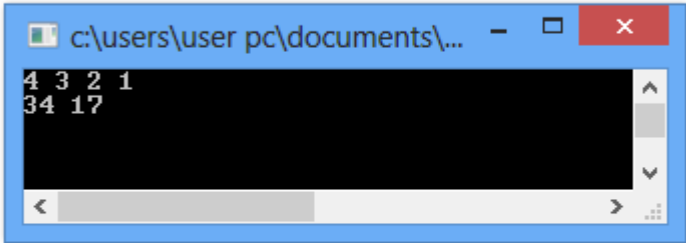
чаще всего применяется для приведения **void*** к некоторому типу.

```
#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;
}
```



6. Явное преобразование приведения типов в C++:

приведения 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).

7. 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*"

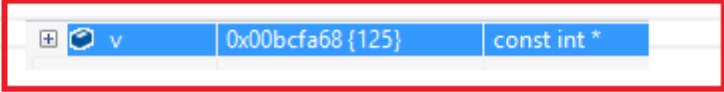
```
#include "stdafx.h"
#include <iostream>
#include <iomanip>

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

    int m = k/5;

    const int* v = &k;

    return 0;
}
```



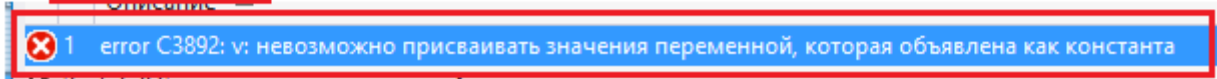
```
#include "stdafx.h"
#include <iostream>
#include <iomanip>

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

    int m = k/5;

    const int* v = &k;

    (*v) = 15;
}
```



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.

8. 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;
}
```

9. Область видимости переменных в C++:

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

10. 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.

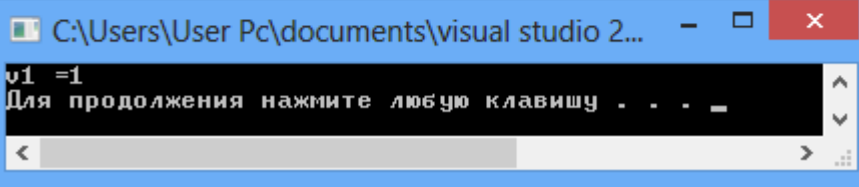
В зависимости от области действия переменная может быть локальной или глобальной.


```
#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;
}
```



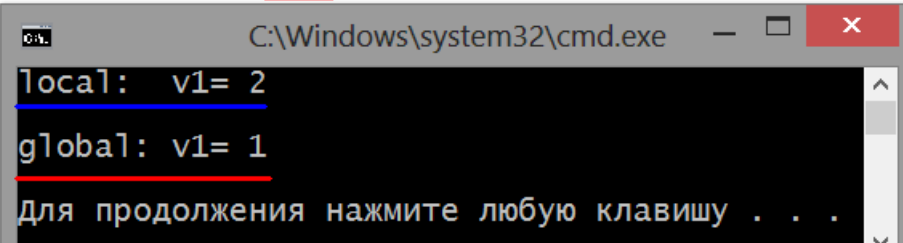
```
#include "stdafx.h"
#include <iostream>

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;
}
```



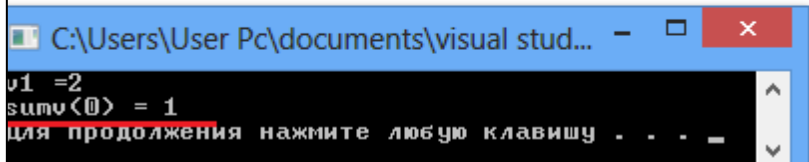
```
#include "stdafx.h"
#include <locale>
#include <iostream>

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;}
```



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

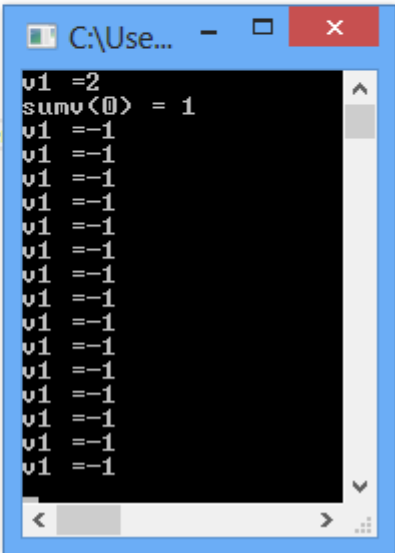
int v1 = 1;
int sumv(int k);

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

    };

    system("pause");
    return 0;
}

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

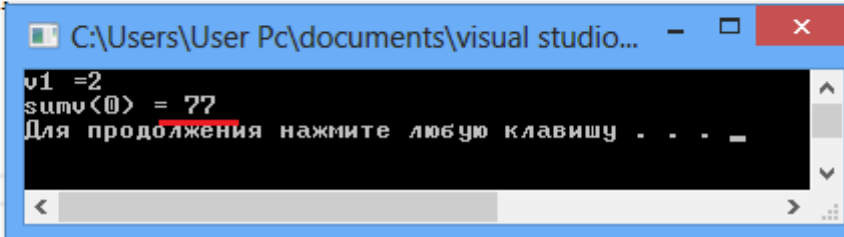


```
#include "stdafx.h"
#include <locale>
#include <iostream>

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(77)<<std::endl;
    system("pause");
    return 0;
}

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



11. 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>.

12. Пространство имен в C++:

namespace, using, псевдонимы пространства имен.

Для доступа к определенным внутри области объектам, используется **оператор разрешения видимости «::»**.

```

#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;
}

```

```

#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
 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;
}

```

```

#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;
}

```

```

#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};

}

```

#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;
}

```

```

#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,
    system( pause );
    return 0;
}

```

```

#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<< "X::c = "<<X::c<<" X::x = "<<X::x<<" X::func_mulA(5.0, 5.5) = " <<X::func_mulA(5.0, 5.5) <<std::endl;
    std::cout<< "Y::b = "<<Y::b<<" Y::y = "<<Y::y<<" Y::func_mulB(5.0, 5.5) = " <<Y::func_mulB(5.0, 5.5) <<std::endl;
    system( pause );
    return 0;
}

```