Программирование

Часть 2
Основы программирования

3. Основы программирования
3.1. Языки программирования

Программа = Структуры данных + Алгоритмы

- **СТРУКТУРА ДАННЫХ** – способ представления данных в программе для компьютера
- **АЛГОРИТМ** – заранее заданная последовательность однозначно трактуемых команд для получения решения задачи за конечное число шагов
- **ПРОГРАММА** - описание структур данных и алгоритма решения задачи на языке программирования, автоматически переводимое на язык машинных команд конкретной ЭВМ помощи транслятора (интерпретатора).
- **ЯЗЫК ПРОГРАММИРОВАНИЯ** — формальная знаковая система, предназначенная для описания структур данных и алгоритмов в форме, пригодной для дальнейшей компиляции (интерпретации) и исполнения на компьютере. Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы.

3. Основы программирования
3.1. Языки программирования

Уровень языка	Примеры
Высокий	Ada
	Modula-2
	Pascal
	COBOL
	FORTRAN
	BASIC
Средний	Java
	C#
	C++
	C
	Forth
Низкий	Macro-assembler
	Assembler

- **Использует концепцию типа данных**
- **Структурирован**
- **Машинонезависим**

- **Свободные преобразования типов**
- **Манипуляции с адресами, битами и байтами**
- **Не контролирует ошибки на этапе выполнения программы**

3. Основы программирования
3.1. Языки программирования

Структурированные языки:
- обособление кода и данных
- использование стандартных управляющих структур
- свободное форматирование текста программы

Блочно структурированные языки:
- возможность объявления вложенных функций

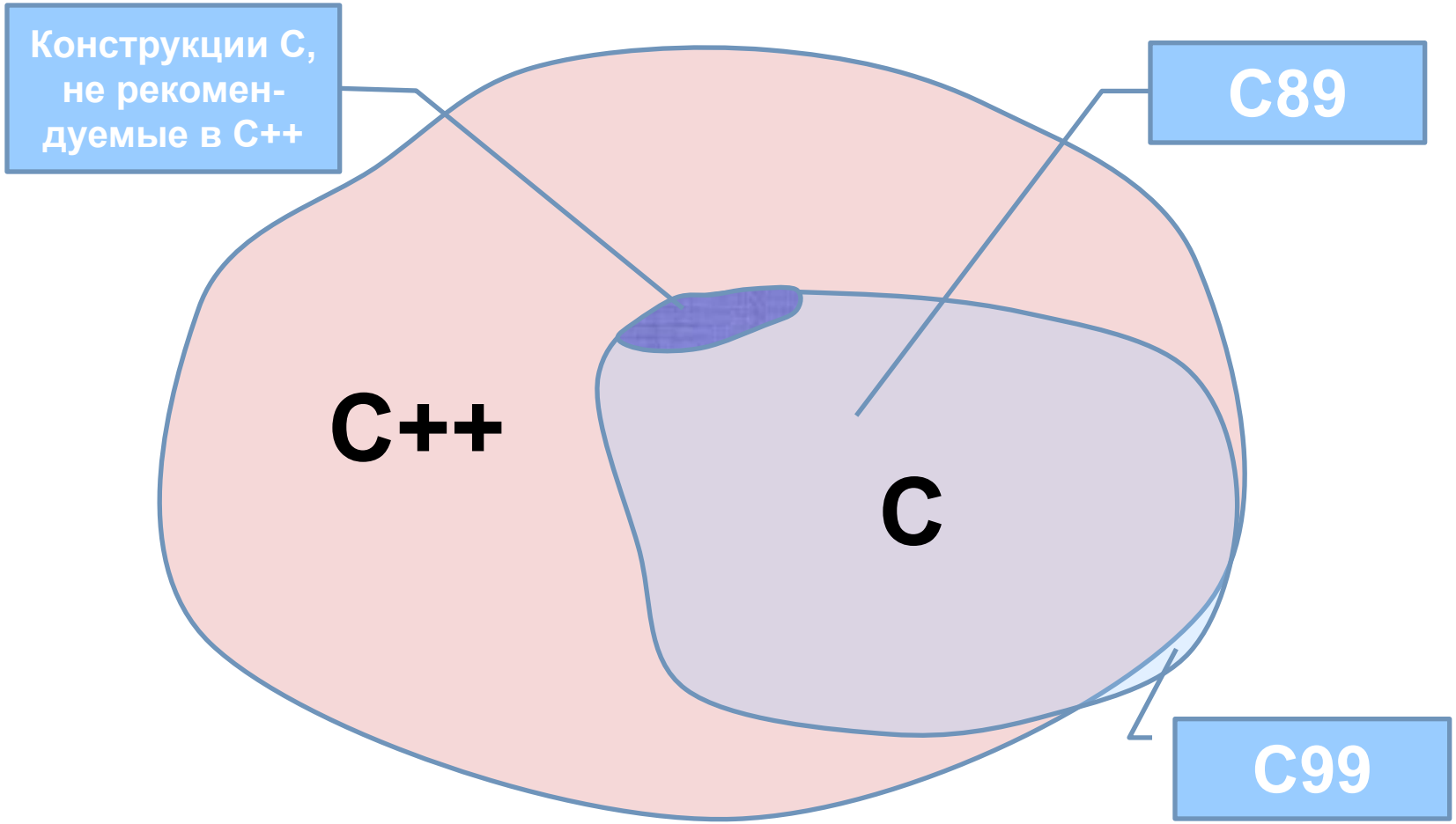
Структурированность	Примеры
Блочно структурированные	Modula-2
	Pascal
Структурированные	C++
	C
Неструктурированные	COBOL
	FORTRAN
	BASIC

3. Основы программирования
3.1. Языки программирования

- http://www.tiobe.com/tpci.htm The TIOBE Programming Community index gives an indication of the popularity of programming languages. The index is updated once a month. The ratings are based on the world-wide availability of skilled engineers, courses and third party vendors. The popular search engines Google, MSN, and Yahoo! are used to calculate the ratings.

Position Aug 2009	Position 2007	Delta in Position	Programming Language	Ratings 2009	Delta 2007
1	1	=	Java	19.5%	-1.5%
2	2	=	C	17.2%	+0.9%
3	3	=	C++	10.5%	0.0%
5	4	↓	(Visual) Basic	8.4%	-1.3%
4	5	↑	PHP	9.4%	+0.8%
8	6	↓↓	Perl	4.0%	-1.30%
7	7	=	C#	4.4%	+0.7%
6	8	↑↑	Python	4.4%	+1.4%
9	9	=	JavaScript	2.8%	+0.2%
10	10	=	Ruby	2.5%	+0.5%

3. Основы программирования
3.1. Языки программирования



3. Основы программирования
3.2. Простейшая программа на C++

```
// Эта программа рассчитывает скорость,
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
	const float g=9.8;
	float h;
	cout << "Please, enter the value of height (m): ";  cin >>h;
	float v=sqrt(2.0*g*h);
	cout << "Calculated value of velocity (m/s) is  " << v << endl;
	_getch();
	return 0;
}
```

3. Основы программирования
3.3. Комментарии

```
// Эта программа рассчитывает скорость,
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
      const float g=9.8;
      float h;
      cout << "Please, enter the value of height (m): ";  cin >>h;
      float v=sqrt(2.0*g*h);
      cout << "Calculated value of velocity (m/s) is  " << v << endl;
      _getch();
      return 0;
}
```

Комментарии

3. Основы программирования
3.3. Комментарии

```
// Эта программа рассчитывает скорость,
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>          // В stdafx включены conio.h math.h и iostream
using namespace std;         // Используем пространство имен std
int main()                   // Головная программа
{                            // Начало тела функции main
    const float g=9.8;       // Ускорение свободного падения
    float h;                 // Высота
    cout << "Please, enter the value of height (m): ";  cin >>h;    // Ввод высоты
    float v=sqrt(2.0*g*h);   // Расчет скорости из закона сохранения энергии
    cout << "Calculated value of velocity (m/s) is  " << v << endl; //Вывод скорости
    _getch();   /* Ждем ввода любого символа */
    return 0;                // Возвращаем в ОС код возврата 0
}                            // Конец тела функции main
```

Пример излишних коммента-риев

3. Основы программирования
3.3. Комментарии

```
// Эта программа рассчитывает скорость,
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
    const float g=9.8;        // Ускорение свободного падения
    float h;                  // Высота
    cout << "Please, enter the value of height (m): ";  cin >>h;
    float v=sqrt(2.0*g*h);    // Расчет скорости из закона сохранения энергии
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();   // Ждем ввода любого символа перед завершением
    return 0;
}
```

Пример разумных коммента-риев

3. Основы программирования
3.4. Директивы препроцессора

```
// Эта программа рассчитывает скорость,
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
    const float g=9.8;
    float h;
    cout << "Please, enter the value of height (m): ";  cin >>h;
    float v=sqrt(2.0*g*h);
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();
    return 0;
}
```

Директива препроцессора: включить заголовочный файл

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently

#pragma once

#define WIN32_LEAN_AND_MEAN  // Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>

// TODO: reference additional headers your program requires here
#include <conio.h>
#include <cmath>
#include <iostream>
```

Пример заголовочного файла stdafx.h

3. Основы программирования
3.5. Пространство имен

```
// Эта программа рассчитывает скорость
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
      const float g=9.8;
      float h;
      cout << "Please, enter the value of height (m): ";  cin >>h;
      float v=sqrt(2.0*g*h);
      cout << "Calculated value of velocity (m/s) is  " << v << endl;
      _getch();
      return 0;
}
```

Директива using: использовать пространство имен std

3. Основы программирования
3.5. Пространство имен

```
// Эта программа рассчитывает скорость
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>


int main()
{
      const float g=9.8;
      float h;
      std::cout << "Please, enter the value of height (m): ";  std::cin >>h;
      float v=sqrt(2.0*g*h);
      std::cout << "Calculated value of velocity (m/s) is  " << v << std::endl;
      _getch();
      return 0;
}
```

Явное указание пространства имен с использованием операции разрешения области действия::

```
// Эта программа рассчитывает скорость
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
    const float g=9.8;
    float h;
    cout << "Please, enter the value of height (m): ";  cin >>h;
    float v=sqrt(2.0*g*h);
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();
    return 0;
}
```

Головная программа: функция main

Функции

- **Функция** – особым образом оформленный фрагмент программы, имеющий собственное имя. Отличительная черта функции – круглые скобки после имени, в которые могут быть заключены аргументы функции.
- Для выполнения функции достаточно указать ее имя, за которым может следовать список аргументов, заключенный в круглые скобки.
- Список аргументов может обеспечивать как передачу данных в функцию, так и возврат значений аргументов из нее.
- Аргументами могут быть константы, переменные и другие объекты программы.
- Результат работы функции возвращается в виде значения этой функции и может использоваться наряду с другими операндами в выражениях.

Функция main

```
int main()
{
    const float g=9.8;
    float h;
    cout << "Please, enter the value of height (m): ";  cin >>h;
    float v=sqrt(2.0 * g * h);
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();
    return 0;
}
```

Имя функции

Функция main

```
int main()
{
    const float g=9.8;
    float h;
    cout << "Please, enter the value of height (m): ";  cin >>h;
    float v=sqrt(2.0 * g * h);
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();
    return 0;
}
```

Тип возвращаемого значения

Возвращаемое значение

Функция main

```
int main()
{
    const float g=9.8;
    float h;
    cout << "Please, enter the value of height (m): ";  cin >>h;
    float v=sqrt(2.0 * g * h);
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();
    return 0;
}
```

Тело функции

Операторные скобки

```
// Эта программа рассчитывает скорость
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
      const float g=9.8;
      float h;
      cout << "Please, enter the value of height (m): ";  cin >>h;
      float v=sqrt(2.0 * g * h);
      cout << "Calculated value of velocity (m/s) is  " << v << endl;
      _getch();
      return 0;
}
```

Библиотечные функции sqrt и _getch

Библиотечные функции

- Описываются в заголовочных файлах, включаемых в программу с помощью директивы #include
- Исходные тексты (если они доступны) хранятся в одноименных файлах с расширением cpp.
- Откомпилированные функции хранятся в объектных (с расширением obj) или библиотечных (с расширением lib) файлах.
- Примеры обращений к встроенным функциям:

```
v = sqrt (2.0 * g * h);          // math.h
_getch();                        // conio.h
```

3. Основы программирования
3.8. Инструкции

```
// Эта программа рассчитывает скорость
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
    const float g=9.8;
    float h;
    cout << "Please, enter the value of height (m): ";  cin >>h;
    float v=sqrt(2.0*g*h);
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();
    return 0;
}
```

Инструкции (statements) – структурные единицы программы на C++

3. Основы программирования
3.8. Инструкции

- **Инструкция (предложение, оператор языка)** – синтакисически правильное предложение, структурная единица программы на C++.
  Примеры инструкций:

```
const float g=9.8;
cout << "Calculated value of velocity (m/s) is  " << v << endl;
return 0;
```

  - **Ключевые слова** (int, long, return, if и др.) – слова, которые имеют предопределенное значение в языке программирования и не могут использоваться для других целей.
  - **Идентификаторы** – слова языка, которые могут использоваться для обозначения имени переменной, имени функции, имени типа или метки инструкции. При записи идентификаторов (имен) объектов можно использовать латинские буквы, цифры, символ подчеркивания. Идентификатор должен отличаться от ключевых слов. Идентификатор не должен начинаться с цифры. Длина – любая, но компилятор различает первые 31 символ.
- **Прописные и строчные буквы считаются разными символами.**
- **Признак окончания оператора** – знак **;**

3. Основы программирования
3.8. Операторы

- **Разделяющие знаки** – пробелы, символы возврата каретки, табуляции, перехода на новую строку и новую страницу – не обрабатываются компилятором и игнорируются. Исключение: директивы препроцессора и строковые константы.
- Разделяющие знаки нельзя вставлять в идентификаторы (имена) объектов (функций, классов, переменных, констант, манипуляторов и т.п.), знаки операций (например, <<), служебные слова (return и др.)
- Если Вы не хотите расположить строковую константу на одной строке, воспользуйтесь тем, что компилятор объединяет две следующие одна за другой строковые константы.

```
// Эта программа рассчитывает скорость,
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
      const
      float    g          =       9.8;      float    h;
      cout
       << "Please, enter”
          “ the value of height (m): ";  cin >>h;        float v=sqrt(2.0*g*h);
      cout << "Calculated value of velocity (m/s) is  " << v << endl;
      _getch
       ();  return 0;  }
```

Можно

3. Основы программирования
3.9. Константы

Константы

- Константа – информационный объект программы, не изменяющийся в процессе ее исполнения.
- Различают неименованные (литералы) и именованные константы
- В качестве литералов в C++ могут использоваться:

целые числа:

десятичные	**12**	**-123**
восьмеричные	**0736**	**-0121**
шестнадцатеричные	**0X2F56**	**-0x2A13B**
вещественные числа	**0.25**	**-56.12e-12f**
логические	**True**	**False**
символы	**‘Ж’**	**‘2’**
строки	**“проба”**	**“124”**
перечислимые	**{red, yellow, green}**	
неопределенный указатель	**NULL**	

3. Основы программирования
3.9. Константы

Символьные литералы

- Значение – символ. Заключаются в одиночные кавычки. Например:

  ‘a’, ‘#’, ‘1’

- Когда компилятор встречает символьную константу, он заменяет ее значением ASCII кода.

Строковые литералы

- Заключаются в двойные кавычки. Например:

  "Please, enter the value of height (m): "

- Компилятор объединяет две следующие одна за другой строковые константы, разделенные любыми символами - разделителями

- Символы и строки могут содержать управляющие последовательности . При их записи они начинаются с символа \ .

3. Основы программирования
3.9. Константы

Обозначения управляющих символов в литералах	Символ	Шестнадцатеричный код символа
\a	Сигнал	007
\b	Возврат	008
\f	Перевод страницы	00D
\n	Перевод в начало следующей строки	00A
\r	Возврат каретки	00C
\t	Горизонтальная табуляция	009
\v	Вертикальная табуляция	00B
\\	Обратная косая черта	05C
\’	Одинарные кавычки (апостроф)	022
\”	Двойные кавычки	027
\0	Ноль-символ	000
\DDD	Восьмеричный код символа	
\xDDD	Шестнадцатеричный код символа	

3. Основы программирования
3.9. Константы

- Тип литерала неявно определяется его значением
- Каждому литералу в программе соответствует область оперативной памяти ЭВМ, в которой хранится значение этого литерала.
- Именованную константу можно описать, присвоив ей идентификатор (имя), который можно будет затем использовать в программе вместо того, чтобы непосредственно записывать значение константы.
- **Когда следует обязательно использовать именованные константы:**
  - **для задания параметров, управляющих размером структур данных (массивов и др.), числом итераций в циклах, и других, изменение которых может потребоваться при отладке или модернизации программы;**
  - **для обозначения часто встречающихся в программе постоянных величин.**
  - **при использовании констант, имеющих общеупотребительные обозначения**

3. Основы программирования
3.9. Константы

Примеры использования именованных констант:

```
…
const  unsigned int  ar_size=10;

typedef  float   myarray  [ar_size];
typedef  float  mymatrix  [ar_size] [ar_size];
…
myarray  ar_1;
mymatrix  matr_1, matr_2;
…
for (int i=0; i<=(ar_size-1); i++)  matr_1 [i][i]=1;

const  g=9.8;
```

Впрочем, в C++ именованные константы – не более, чем переменные, значения которых нельзя изменять

3. Основы программирования
3.10. Переменные

```
// Эта программа рассчитывает скорость
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
      const float g=9.8;
      float h;
      cout << "Please, enter the value of height (m): ";  cin >>h;
      float v=sqrt(2.0 * g * h);
      cout << "Calculated value of velocity (m/s) is  " << v << endl;
      _getch();
      return 0;
}
```

Переменные h и v

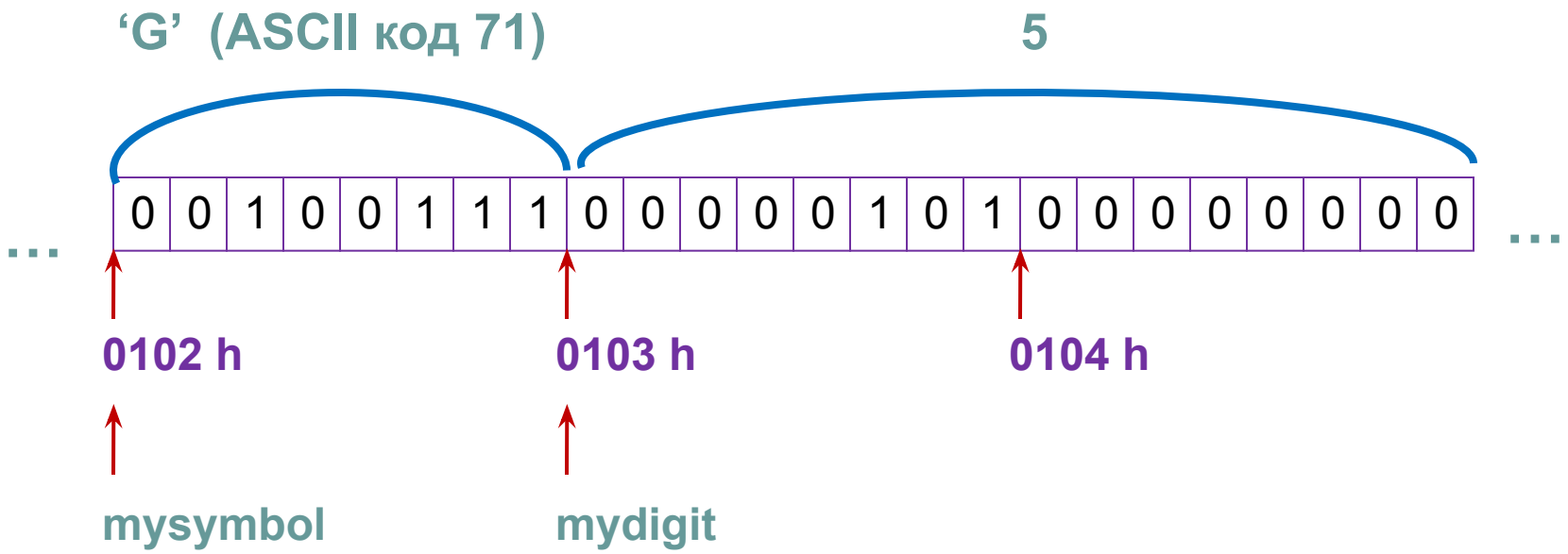
3. Основы программирования
3.10. Переменные

- **Переменная – информационный объект программы, предназначенный для хранения значений, которые могут изменяться в процессе исполнения программы.**
- Переменная имеет идентификатор (имя) по которому в программе осуществляется доступ к содержимому и адресу переменной.
- Каждой переменной в программе соответствует область оперативной памяти ЭВМ, в которой хранится значение этой переменной.
- **Каждая переменная относится к тому или иному типу данных, который определяет:**
  - размер отведенной для переменной области памяти;
  - множество возможных значений, которые может принимать переменная, то есть как интерпретировать информацию (последовательность двоичных чисел), записанных в этой области памяти;
  - Множество операций, которые допустимы над переменной данного типа

3. Основы программирования
3.10. Переменные

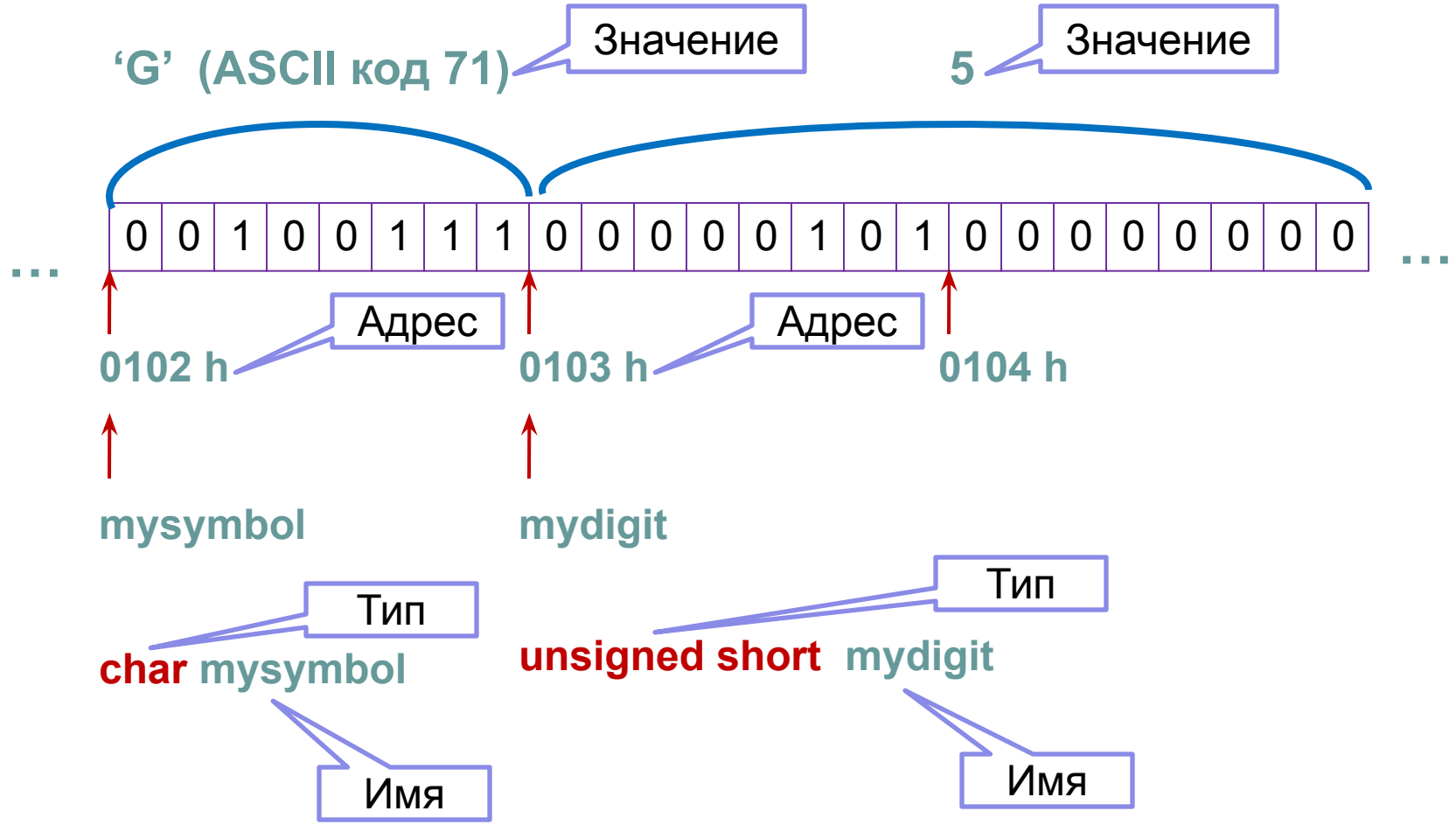
Идея: символическое имя начального байта области памяти, где размещаются данные

3. Основы программирования
3.10. Переменные



char mysymbol; **unsigned short** mydigit;

Идея: символическое имя начального байта области памяти, где размещаются данные + размер области + способ интерпретации этих данных


'G' (ASCII код 71)
Значение
5
Значение
0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0
Адрес
0102 h
Адрес
0103 h
0104 h
mysymbol
mydigit
Тип
char mysymbol
Имя
Тип
unsigned short mydigit
Имя

3. Основы программирования
3.10. Переменные

- **Перед тем, как использовать переменную, ее нужно объявить или определить.**
- **Объявление -** введение нового идентификатора в программе. Объявление указывает тип, к которому будет относиться объект (переменная, функция и т.п.), которому присваивается идентификатор. С помощью объявления обычно также можно ввести новый идентификатор для типа данных.
- Объявления, в отличие от определений не определяют сущность, соответствующую идентификатору. В частности, объявление какого-либо объекта не осуществляет выделение памяти для этого объекта.
- **Определение -** объявление идентификатора, которое дополнительно определяет сущность, соответствующую данному идентификатору (например, для переменной при ее определении выделяется требуемый объем оперативной памяти).

3. Основы программирования
3.10. Переменные

- В C++ объявление переменной одновременно обычно является ее определением (если переменная не определена извне области программы, где она объявлена со спецификатором extern) .
- Синтаксис определения переменной:

  Тип ИдентПерем;

  Тип ИдентПерем1, ИдентПерем2, ИдентПерем3 …;

  Например: **float h;**

  int a, b, c;
- Приветствуется инициализация переменных при их описании

  Тип ИдентПерем=Знач;

  Тип ИдентПерем1=Знач1, ИдентПерем2=Знач2…;

  Например: **int a=3, b=4;**
- Или так

  Тип ИдентПерем (Знач);

  Тип ИдентПерем1 (Знач1), ИдентПерем2 (Знач2)…;

  Например: **double c (13.0) , d (12.4e-4);**

```
// Эта программа рассчитывает скорость,
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
    const float g=9.8;
    float h, v;

    cout << "Please, enter the value of height (m): ";  cin >>h;
    v=sqrt(2.0*g*h);
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();
    return 0;
}
```

Стиль 1 (C):
Описания в начале функции

3. Основы программирования
3.10. Переменные

```
// Эта программа рассчитывает скорость,
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
    cout << "Please, enter the value of height (m): "; float h; cin >>h;
    const float g=9.8;
    float v=sqrt(2.0*g*h);
    cout << "Calculated value of velocity (m/s) is  " << v << endl;
    _getch();
    return 0;
}
```

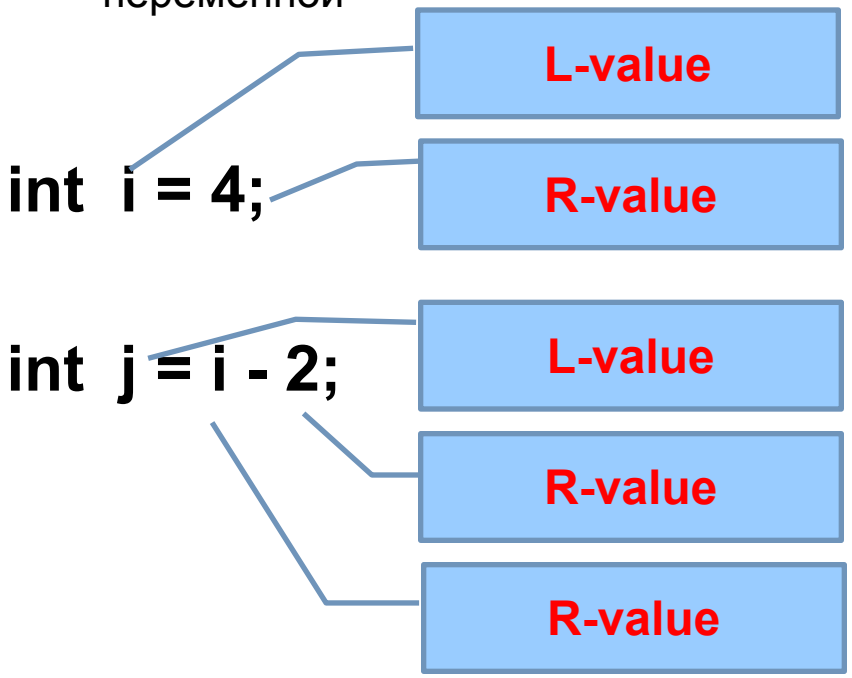
Стиль 2 (C++):
Описания приближены к месту первого использования

L-value и R-value

- Данные в C++ могут представляться неименованными константами (литералами) и переменными (именованные константы можно рассматривать как частный случай переменных, значения которых нельзя изменять в ходе выполнения программы.
- Литералы и переменные хранят свои значения в областях памяти. Это хранимое **значение данных называется R-value** (от Read value – значение которое можно прочитать)
- Переменные (в отличие от литералов) именуют область памяти, в которой хранится значение - R-value. Таким образом, имя переменной предоставляет доступ к области памяти, где хранится значение.
- **Значение адреса области памяти, где хранится значение переменной, называется L-value** (от Location value – значение, определяющее местоположение).
- Так как L-value определяет адрес, то по этому адресу может быть записано новое значение переменной.

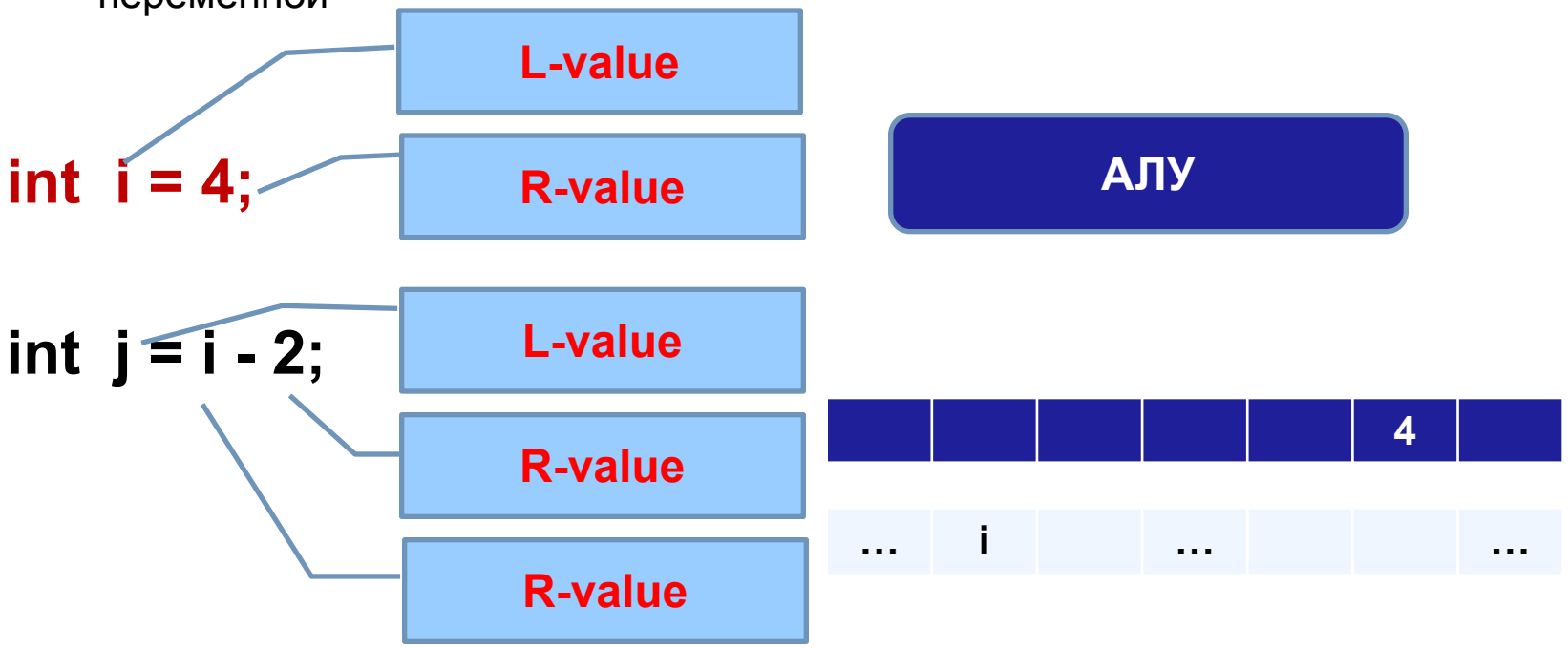
L-value и R-value

- Понятия L-value и R-value вводятся для того, чтобы точнее определить, какие значения могут быть операндами (аргументами) операций и какое значение получается в результате вычисления выражений.
- Рассмотрим, например, операторы объявления и инициализации переменной



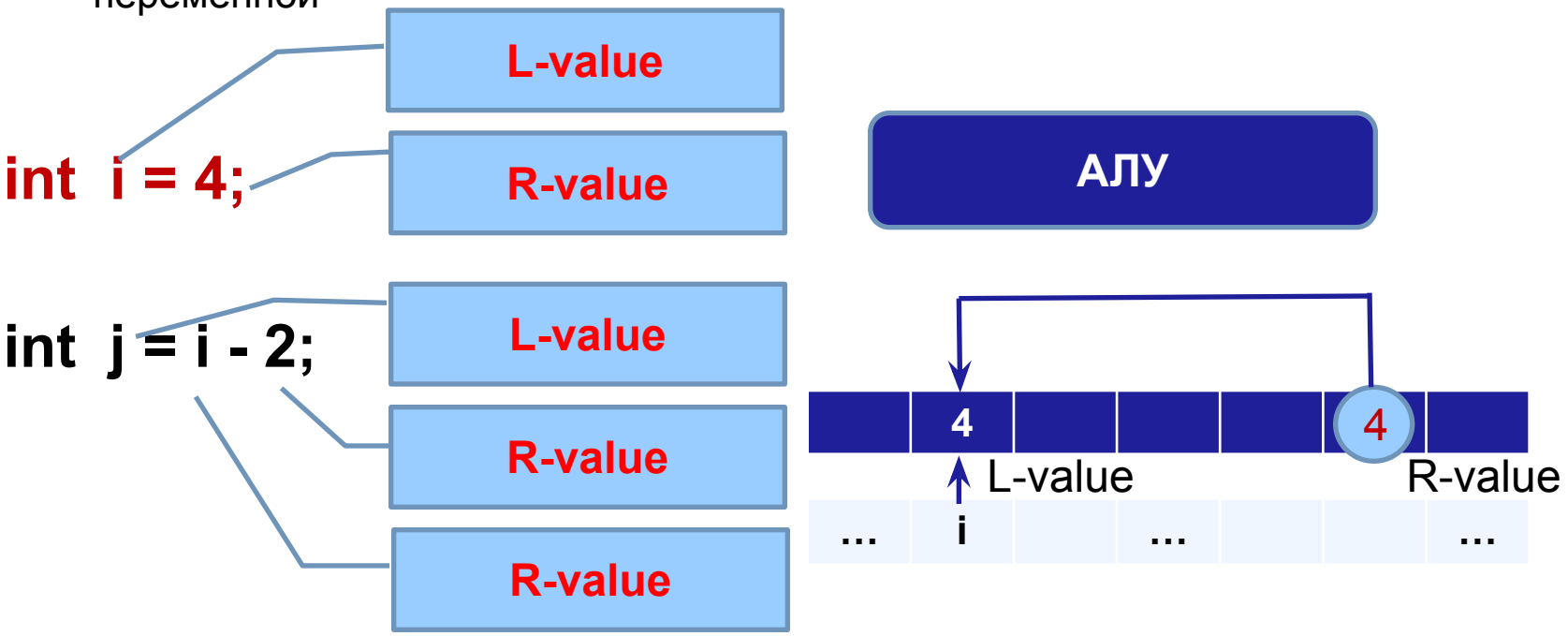
L-value и R-value

- Понятия R-value и R-value вводятся для того, чтобы точнее определить, какие значения могут быть операндами операций и какое значение получается в результате вычисления выражений.
- Рассмотрим, например, операторы объявления и инициализации переменной



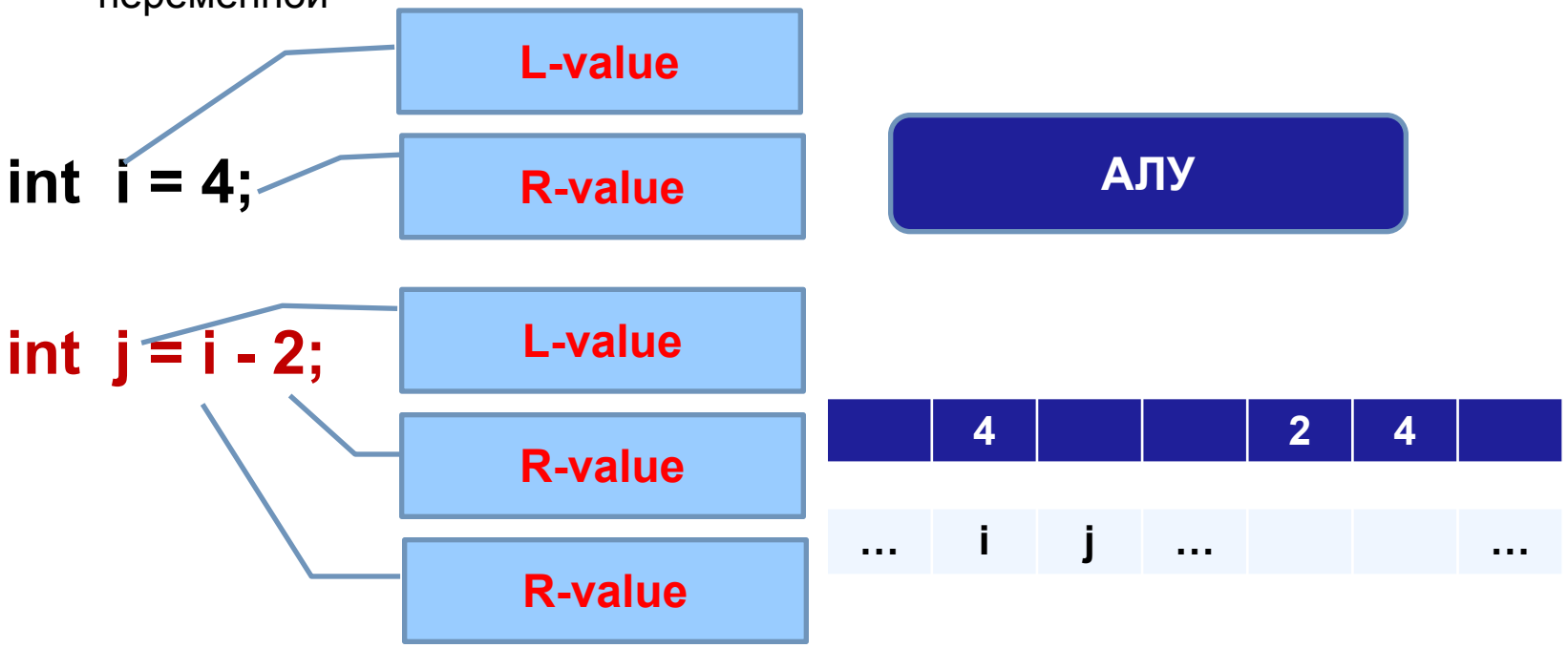
L-value и R-value

- Понятия R-value и R-value вводятся для того, чтобы точнее определить, какие значения могут быть операндами операций и какое значение получается в результате вычисления выражений.
- Рассмотрим, например, операторы объявления и инициализации переменной



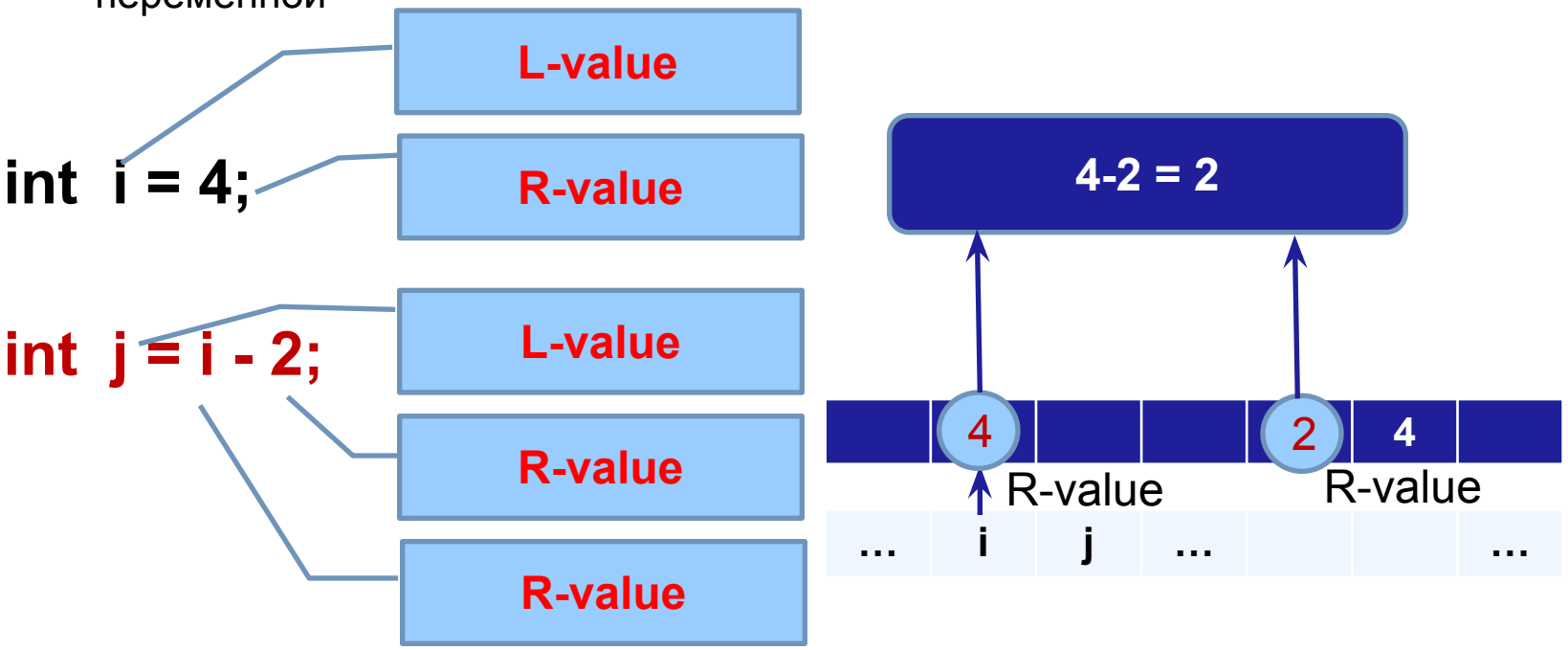
L-value и R-value

- Понятия R-value и R-value вводятся для того, чтобы точнее определить, какие значения могут быть операндами операций и какое значение получается в результате вычисления выражений.
- Рассмотрим, например, операторы объявления и инициализации переменной



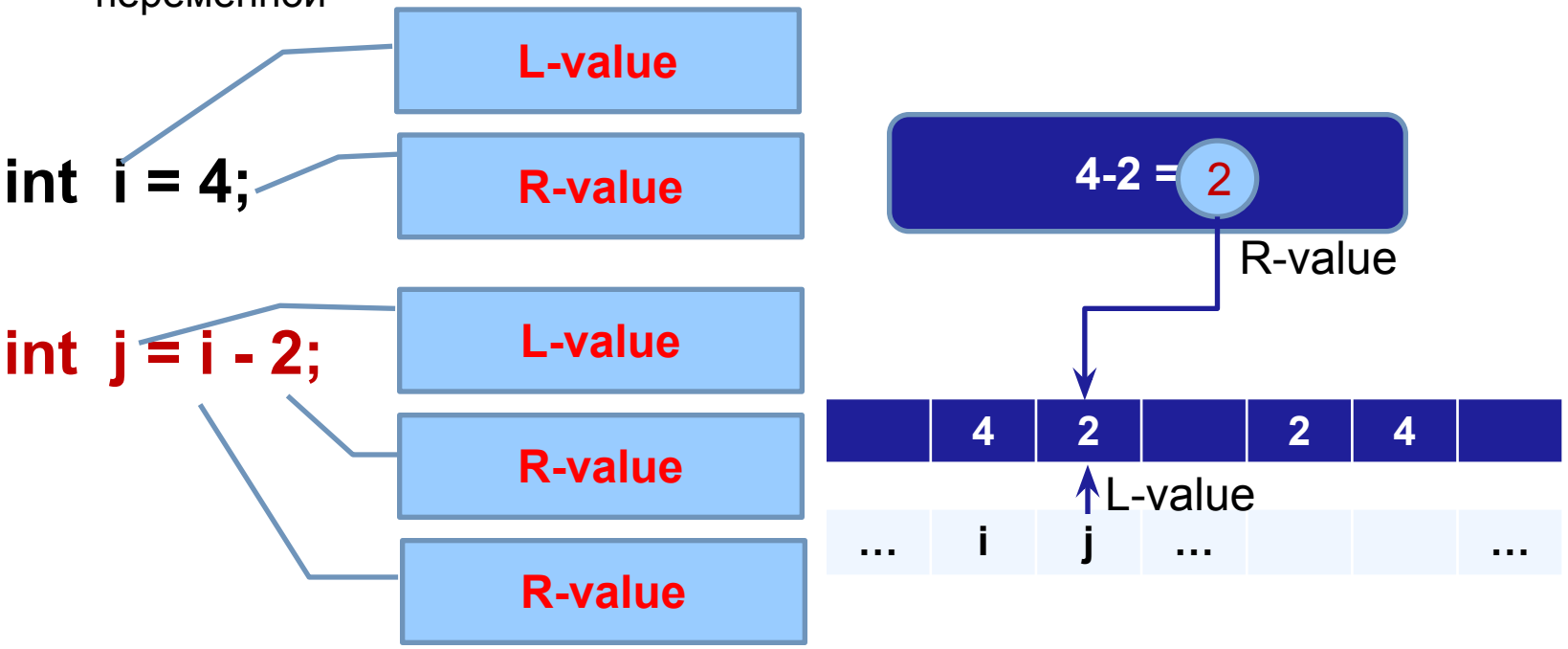
L-value и R-value

- Понятия R-value и R-value вводятся для того, чтобы точнее определить, какие значения могут быть операндами операций и какое значение получается в результате вычисления выражений.
- Рассмотрим, например, операторы объявления и инициализации переменной



L-value и R-value

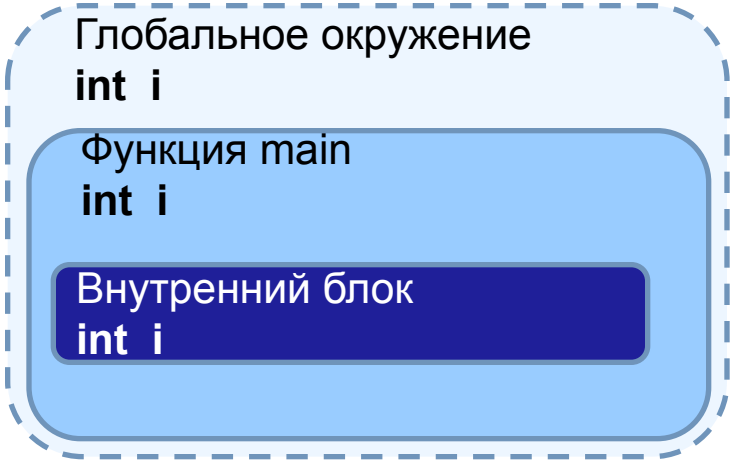
- Понятия R-value и R-value вводятся для того, чтобы точнее определить, какие значения могут быть операндами операций и какое значение получается в результате вычисления выражений.
- Рассмотрим, например, операторы объявления и инициализации переменной



Область видимости переменных

- Переменные доступны в том блоке программы, в котором они объявлены.
- Блок ограничивается фигурными скобками. Например, тело функции main (как и любой другой функции) является блоком.
- Внутри функции может быть введено сколько угодно блоков, и внутри каждого из них можно описывать локальные переменные, доступные только внутри этого блока.
- Если во вложенном блоке описана переменная с тем же именем, что и в охватывающем, внутри блока будет доступна описанная в нем локальная переменная. После завершения вложенного блока, в охватывающем блоке будет доступна описанная в нем переменная (и ее значение).
- Если переменная описана вне функций, она является глобальной и доступна из любой части программы.
- Доступ к глобальной переменной может быть утрачен при объявлении одноименной локальной переменной
- Для доступа к одноименной глобальной переменной из блока, где описана одноименная локальная переменная, следует использовать оператор разрешения области действия (разрешения области видимости) **::**

Область видимости переменных



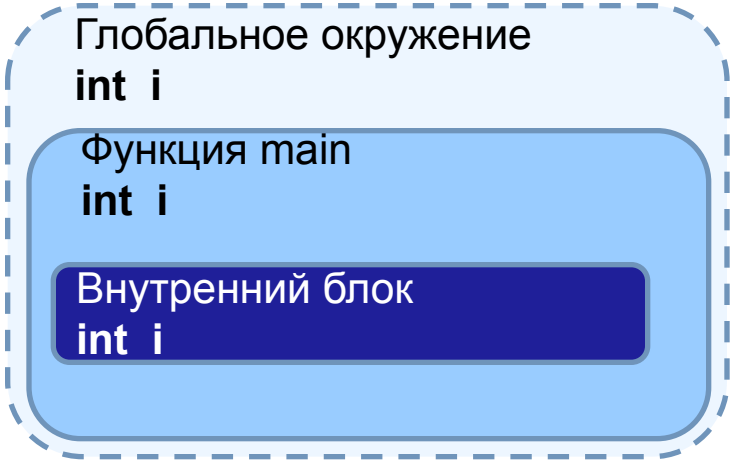

```
#include <stdafx.h>
using namespace std;

int i=2;   // глобальная переменная

int main()
{
        cout << i << endl;                                  //  2
        int i=5;  cout << i << "   " << ::i << endl;                  //  5    2
        {
                cout << i << "   " << ::i << endl;          //  5    2
                int i=10; cout << i << "   " << ::i << endl;          //  10   2
        }
        cout << i << "   " << ::i << endl;                  //  5    2
        return 0;
}
```

	2						
	i	…	…	…	…	…	…

Область видимости переменных



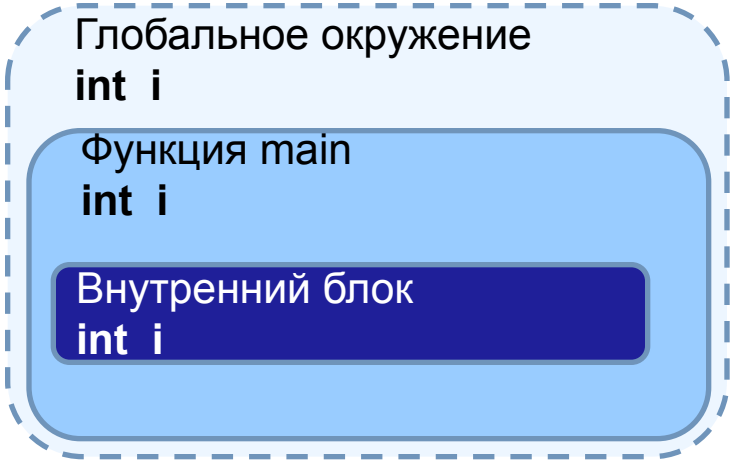

```
#include <stdafx.h>
using namespace std;

int i=2;   // глобальная переменная

int main()
{
        t << i << endl;                                  //  2
    int i=5;  cout << i << "   " << ::i << endl;                  //  5    2
    {
            cout << i << “   “ << ::i << endl;        //  5    2
            int i=10; cout << i << "   " << ::i << endl;         //  10   2
    }
    cout << i << “   “ << ::i << endl;                //  5    2
    return 0;
}
```

	2						
	i	…	…	…	…	…	…

Область видимости переменных



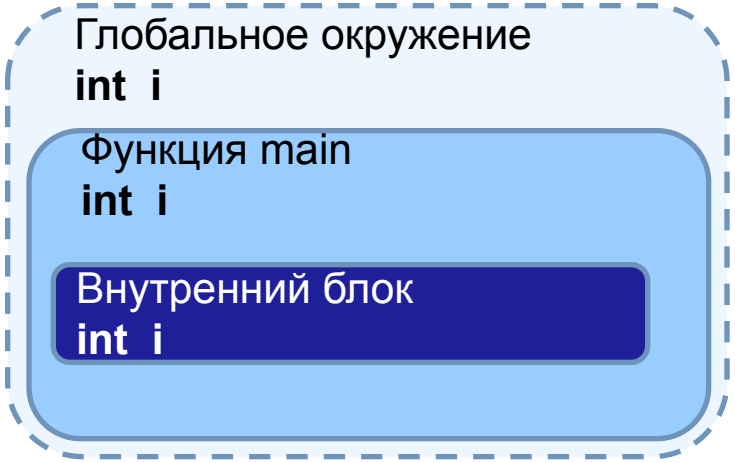

```
#include <stdafx.h>
using namespace std;

int i=2;   // глобальная переменная

int main()
{
        cout << i << endl;                                  // 2
        int i=5;  cout << i << "   " << ::i << endl;                  // 5    2
        {
                cout << i << "   " << ::i << endl;          // 5    2
                int i=10; cout << i << "   " << ::i << endl;          // 10   2
        }
        cout << i << "   " << ::i << endl;                 // 5    2
        return 0;
}
```

	2		5				
	::i	…	i	…	…	…	…

Область видимости переменных



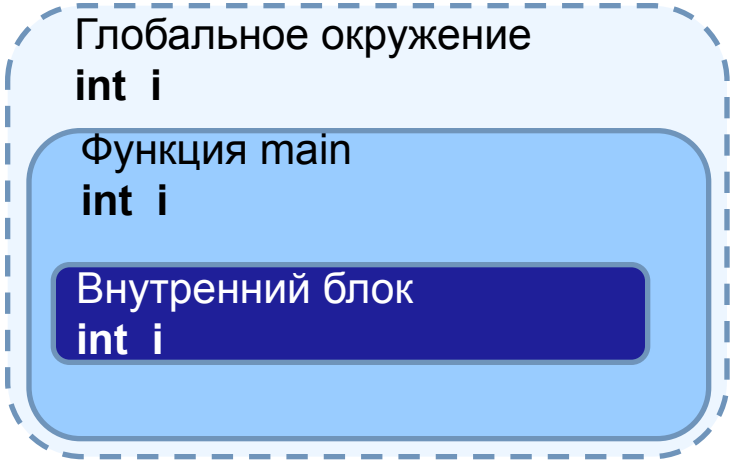

```
#include <stdafx.h>
using namespace std;

int i=2;   // глобальная переменная

int main()
{
        cout << i << endl;                                  // 2
        int i=5;  cout << i << "   " << ::i << endl;              // 5   2
        {
                cout << "   " << ::i << endl;          // 5   2
                int i=10; cout << i << "   " << ::i << endl;      // 10   2
        }
        cout << i << "   " << ::i << endl;              // 5   2
        return 0;
}
```

	2		5				
	::i	…	i	…	…	…	…

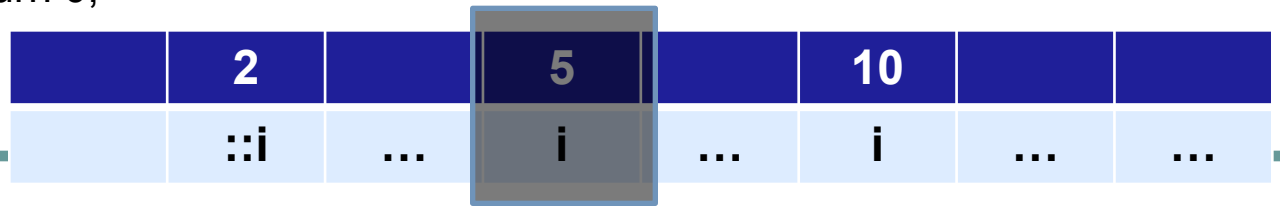
Область видимости переменных



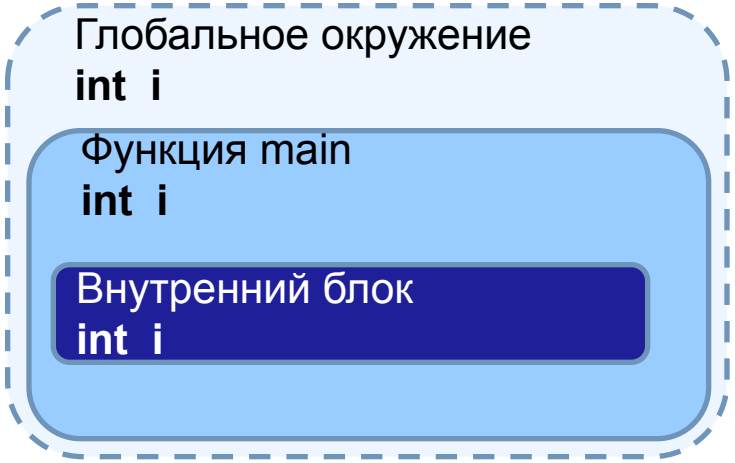

```
#include <stdafx.h>
using namespace std;

int i=2;   // глобальная переменная

int main()
{
        cout << i << endl;                                  //  2
        int i=5;  cout << i << "   " << ::i << endl;                //  5    2
        {
                cout << i << "   " << ::i << endl;          //  5    2
                int i=10; cout << i << "   " << ::i << endl;        //  10   2
        }
        cout << i << "   " << ::i << endl;                  //  5    2
        return 0;
}
```



Область видимости переменных



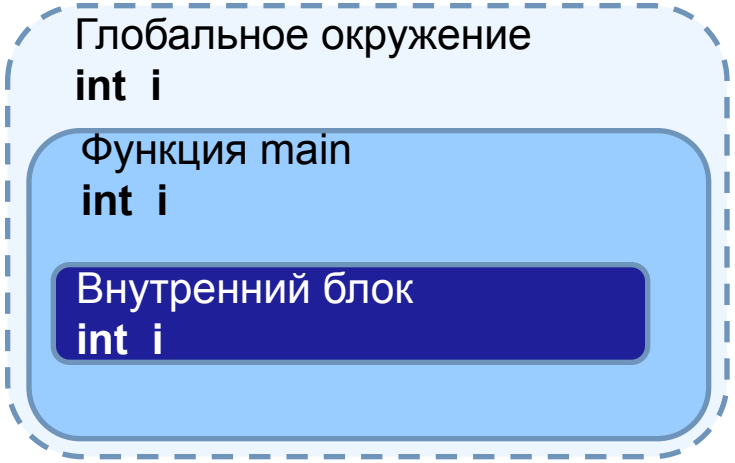

```
#include <stdafx.h>
using namespace std;

int i=2;   // глобальная переменная

int main()
{
        cout << i << endl;                                  // 2
        int i=5;  cout << i << "   " << ::i << endl;                  // 5   2
        {
                cout << i << "   " << ::i << endl;          // 5   2
                int i=10; cout << i << "   " << ::i << endl;          // 10   2
        }
        t << i << "   " << ::i << endl;                    // 5   2
        return 0;
}
```

	2		5				
	::i	…	i	…		…	…

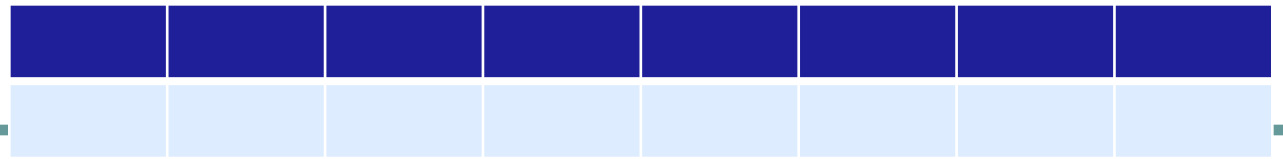
Область видимости переменных




```
#include <stdafx.h>
using namespace std;

int i=2;   // глобальная переменная

int main()
{
        cout << i << endl;                                  //  2
        int i=5;  cout << i << "   " << ::i << endl;                  //  5    2
        {
                cout << i << "   " << ::i << endl;          //  5    2
                int i=10; cout << i << "   " << ::i << endl;          //  10   2
        }
        cout << i << "   " << ::i << endl;                  //  5    2
        return 0;
}
```



3. Основы программирования
3.11. Выражения, операции и операторы

```
// Эта программа рассчитывает скорость
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
      const float g=9.8;
      float h;
      cout << "Please, enter the value of height (m): ";  cin >>h;
      float v=sqrt(2.0 * g * h);
      cout << "Calculated value of velocity (m/s) is  " << v << endl;
      _getch();
      return 0;
}
```

Операции и операторы

3. Основы программирования
3.11. Выражения, операции и операторы

- Любая комбинация переменных, констант, функций и операций, приводящая к вычислению некоторого значения, называется **выражением**:

```
sqrt(2.0 * g * h)
2.0 * g * h
g * h
```

- Выражения сами могут входить в состав других выражений.
- **Операции** – действия над объектами программы (переменными, константами, выражениями, структурами данных, объектами и др.), задаваемые специально определенными символами - **операторами**. Объекты, над которыми производятся операции называются **операндами**.
- В зависимости от числа требуемых операндов различают **унарные, бинарные и тернарные** операции

3. Основы программирования
3.12. Концепция типа данных

- Каждый объект программы, предназначенный для хранения данных или вырабатывающий данные (константа, переменная, выражение, функция) обычно относится к определенному типу.

- Тип данных **определяется набором возможных значений**, которые может принимать или вырабатывать объект программы (переменная, выражение, константа, функция и др.), относящийся к этому типу, **и совокупностью операций, определенных над этими значениями.**

3. Основы программирования
3.12. Концепция типа данных

- **Концепция типа данных основывается на следующих положениях:**
  1. Любой тип данных определяет множество значений, к которому принадлежит константа, которые может принимать переменная или выражение или вырабатывать операция или функция.
  2. Каждая операция или функция требует аргументов фиксированного типа и выдает результат фиксированного типа. Если операция допускает аргументы нескольких типов, то тип результата можно определить по специальным правилам языка.
- **Статическая типизация** (Fortran, Pascal, C, C++)**:**
  3. Тип значения, задаваемого константой, переменной или выражением, можно определить по их виду или описанию и остается неизменным для переменных.
- **Динамическая типизация** (Python, Ruby, Perl)**:**
  3. Тип значения, задаваемого константой, переменной или выражением определяется присвоенным или выработанным им значением в момент присваивания (выработки), может быть определен по их значению и для переменных изменен в процессе выполнения программы .

3. Основы программирования
3.12. Концепция типа данных

- **В большинстве случаев новые типы данных определяются с помощью ранее определенных типов данных.**

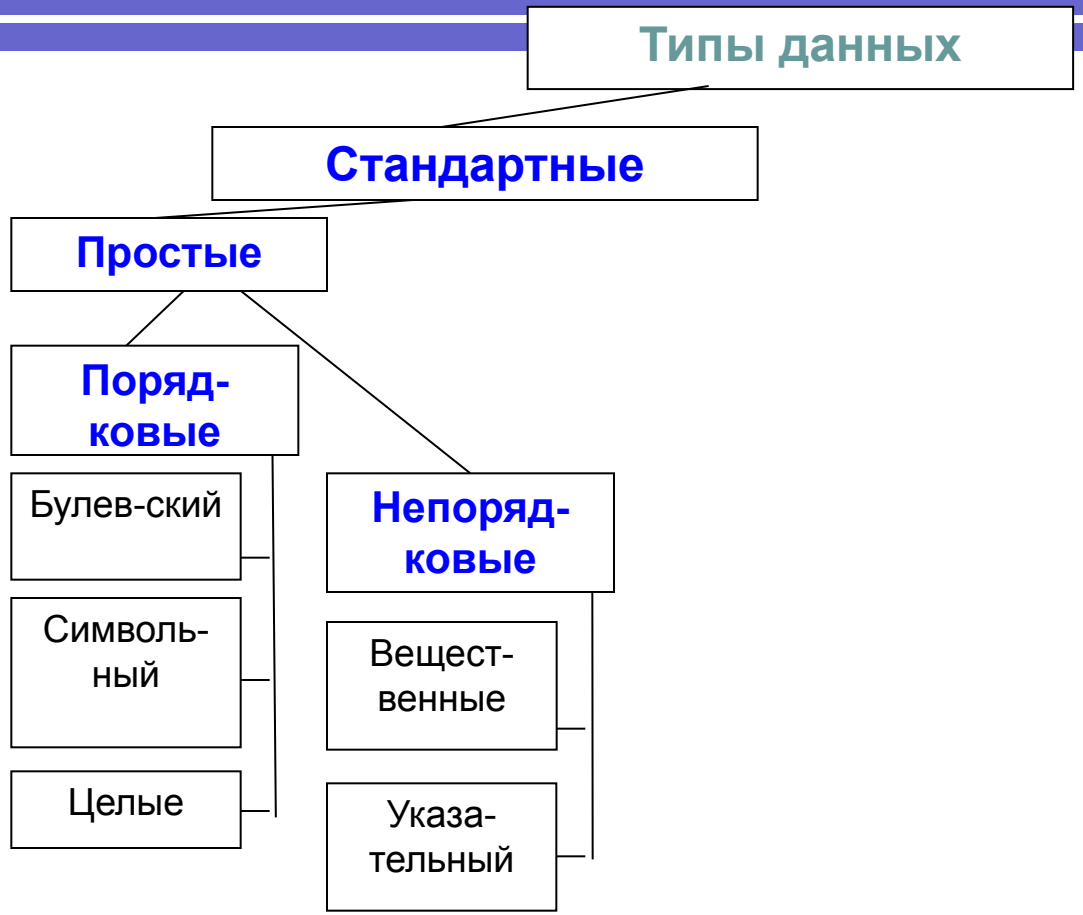
- Значения, принадлежащие к такому типу, обычно представляют собой совокупности ***значений компонент***, принадлежащих к определенным ранее ***типам компонент***. Такие составные значения называются **структурированными.**

- Если значение имеет всего одну компоненту, принадлежащую определенному ранее типу, то этот тип называется **базовым** или **простым**.

- Средства, которыми должен обладать язык программирования:
  1. Стандартные предопределенные типы данных (числовые, логические, символьные, указательные и др.)
  2. Возможность описания новых простых неструктурированных типов (путем перечисления значений, указания интервалов значений и т.п.)
  3. Наличие нескольких методов структурирования, как минимум, позволяющих строить массивы, структуры (записи), последовательности (файлы).

3. Основы программирования
3.13. Классификация типов данных

3. Основы программирования
3.13. Классификация типов данных



3. Основы программирования
3.14. Простые стандартные типы данных языка C++

Название типа	Нижняя граница диапазона	Верхняя граница диапазона	Точность десятичн . разрядов	Размер в байтах
bool	False	True		1
char	-128	127		1
short short int	-32 768	32 767		2
int long long int	-2 147 483 648	2 147 483 647		4
float	$-3.4*10^{38}$	$3.4+10^{38}$	7	4
double	$-1.7*10^{308}$	$1.7*10^{308}$	15	8
void				
void *				4

3. Основы программирования
3.14. Простые стандартные типы данных языка C++

Название типа	Нижняя граница диапазона	Верхняя граница диапазона	Точность десятичн . разрядов	Размер в байтах
unsigned char	0	255		1
unsigned short	0	65 535		2
unsigned int	0	4 294 067 295		4
unsigned long	0	4 294 067 295		4

Основные виды операций в C++

- **по назначению:**
  - присваивания
  - арифметические
  - сравнения
  - логические
  - побитовые
  - условная
  - взятия адреса и разыменования
- **по количеству операндов:**
  - унарные
  - бинарные
  - тернарная

3. Основы программирования
3.16. Операция присваивания

Операция присваивания

Имя_переменной = Выражение;

- Действие: взять **значение** выражения из правой части (rvalue) и записать его в память **по адресу**, на который ссылается переменная в левой части (lvalue).

```
g=9.8;
v=sqrt (2.0 * g * h);
```

- Правило преобразования типов для операции присваивания: значение правой части (rvalue) преобразовать к типу левой части (rvalue)

Операция присваивания: правила преобразования типов

lvalue	=	rvalue	Возможные потери
char, signed char		unsigned char	Если rvalue > 127, результатом будет отрицательное число

Операция присваивания:
преобразование из unsigned char в signed char

```
int main()
{
        char c=0;  unsigned char cs=239;  int i=cs;
        cout << cs <<"   " << i << endl;    // я  239
        i = c = cs;                 // Обратите внимание !!!
        cout << c <<"   " << i << endl;     // я  -17  ☺  ☹

        return 0;
}
```

Операция присваивания: правила преобразования типов

lvalue =	rvalue	Возможные потери
char, signed char	unsigned char	Если rvalue > 127, результатом будет отрицательное число
unsigned char	char, signed char	Если rvalue < 0, результатом будет положительное число

3. Основы программирования
3.16. Операция присваивания

Операция присваивания:
преобразование из signed char в unsigned char

```
int main()
{
        char c=-30;
        int i=c;
        cout << c <<"   " << i << endl;      // т -30

        unsigned char cs=0;
        i = cs = c;
        cout << cs <<"   " << i << endl;    // т   226 ☺  ☹

        return 0;
}
```

Операция присваивания: правила преобразования типов

lvalue =	rvalue	Возможные потери
char, signed char	unsigned char	Если rvalue > 127, результатом будет отрицательное число
unsigned char	char, signed char	Если rvalue < 0, результатом будет положительное число
char	short, short int	Старшие 8 бит
char	int, long int	Старшие 24 бит
short, short int	int, long int	Старшие 16 бит
short, short int, int, long int	float, double	Дробная часть и, возможно, что-то еще

3. Основы программирования
3.16. Операция присваивания

Операция присваивания: преобразование из float в short

```
int main()
{
        short i = 0; float r = -12.9869e2;
        cout << r << endl;                  // - 1298.69
        i = r;
        cout << i <<"   " <<  endl;          // - 1298  ☺

        r= - 123e8;
        cout << r << endl;                  // - 1.23e+10
        i = r;
        cout << i <<"   " <<  endl;          // 0  !!! ☹

        i = 123e14;
        cout << i <<"   " <<  endl;          // - 16384 ☹
        return 0;
}
```

Операция присваивания: правила преобразования типов

lvalue =	rvalue	Возможные потери
char, signed char	unsigned char	Если rvalue > 127, результатом будет отрицательное число
unsigned char	char, signed char	Если rvalue < 0, результатом будет положительное число
char	short, short int	Старшие 8 бит
char	int, long int	Старшие 24 бит
short, short int	int, long int	Старшие 16 бит
short, short int, int, long int	float, double	Дробная часть и, возможно, что-то еще
float	double	Точность; возможны переполнение или потеря порядка

Операция присваивания: преобразование из double во float

```
int main()
{
        float r = 0; double d= -123e14;
        cout << d << endl;    // -1.23e+016
        r = d;
        cout << r <<"   " <<  endl; // -1.23e+016  ☺

        d = -123e112;
        cout << d << endl;    // -1.23e+114
        r = d;
        cout << r <<"   " <<  endl; // -1.#INF (константа –HUGE_VAL ) ☹

        r = 123e-104;
        cout << r <<"   " <<  endl; // 0  ☹
        return 0;
}
```

Операция присваивания: правила преобразования типов

lvalue =	rvalue	Возможные потери
char, signed char	unsigned char	Если rvalue > 127, результатом будет отрицательное число
unsigned char	char, signed char	Если rvalue < 0, результатом будет положительное число
char	short, short int	Старшие 8 бит
char	int, long int	Старшие 24 бит
short, short int	int, long int	Старшие 16 бит
short, short int, int, long int	float, double	Дробная часть и, возможно, что-то еще
float	double	Точность, возможно переполнение или потеря порядка

Операция присваивания в правой части выражения

- Операция присваивания, как и другие, **вырабатывает значение:** это значение равно значению, присвоенному переменной, находящейся в левой части операции.
- Это делает возможным использование операции присваивания в правой части выражений:

```
int main()
{
        int i=0; double x=0;
        x = ( i = 5.25) + 6.5;
        cout << i << "    " << x << endl;      // 5     11.5  (5+6.5)
        return 0;
}
```

Операция присваивания в правой части выражения

- Правильнее: операция присваивания возвращает указатель на переменную (адрес переменной), находящуюся в его левой части.
- Если этот указатель находится в правой части другого оператора присваивания, то он «отдает» значение переменной.

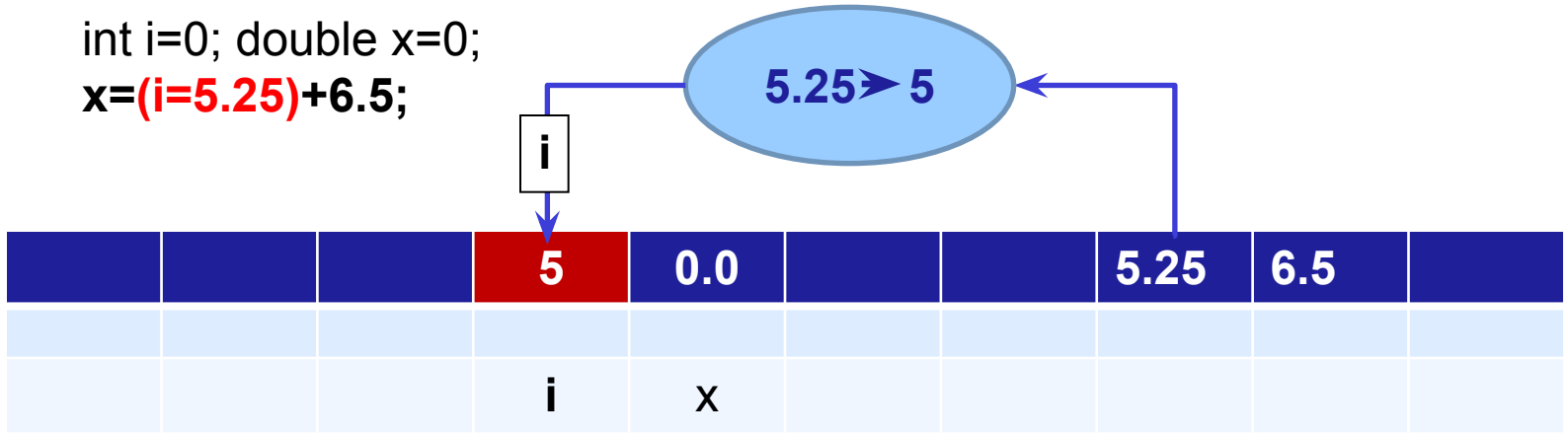
```
int i=0; double x=0;
x=(i=5.25)+6.5;
```

			0	0.0			5.25	6.5	
			i	x					

Операция присваивания в правой части выражения

- Правильнее: операция присваивания возвращает указатель на переменную (адрес переменной), находящуюся в его левой части.
- Если этот указатель находится в правой части другого оператора присваивания, то он «отдает» значение переменной.

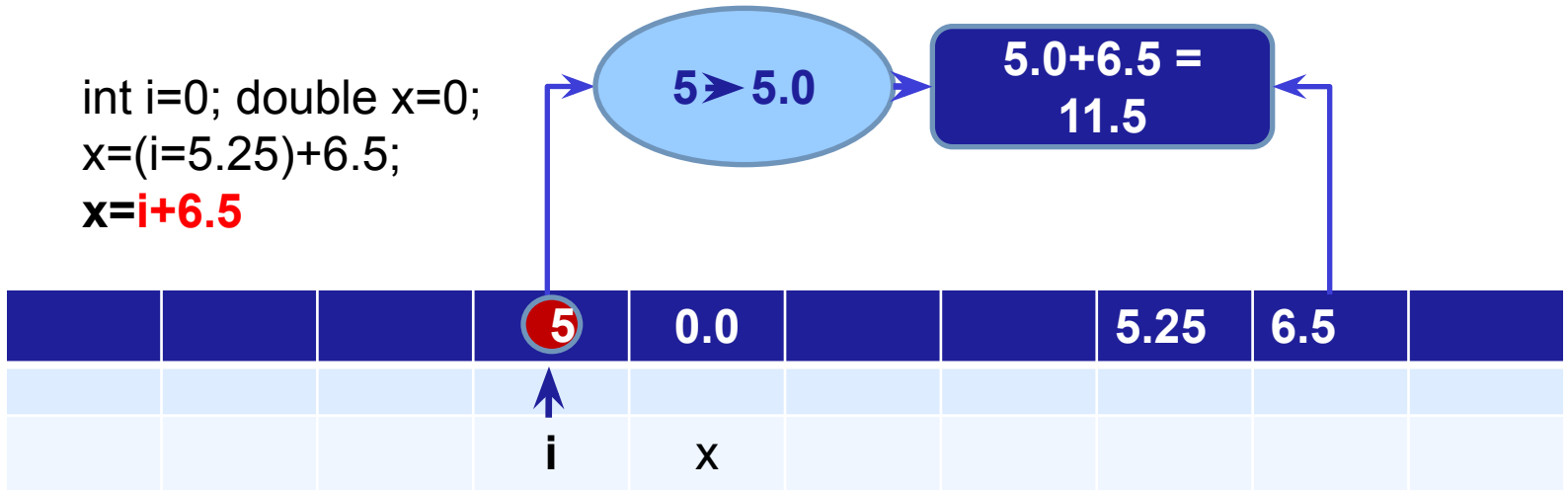
```
int i=0; double x=0;
x=(i=5.25)+6.5;
```



Операция присваивания в правой части выражения

- Правильнее: операция присваивания возвращает указатель на переменную (адрес переменной), находящуюся в его левой части.
- Если этот указатель находится в правой части другого оператора присваивания, то он «отдает» значение переменной.

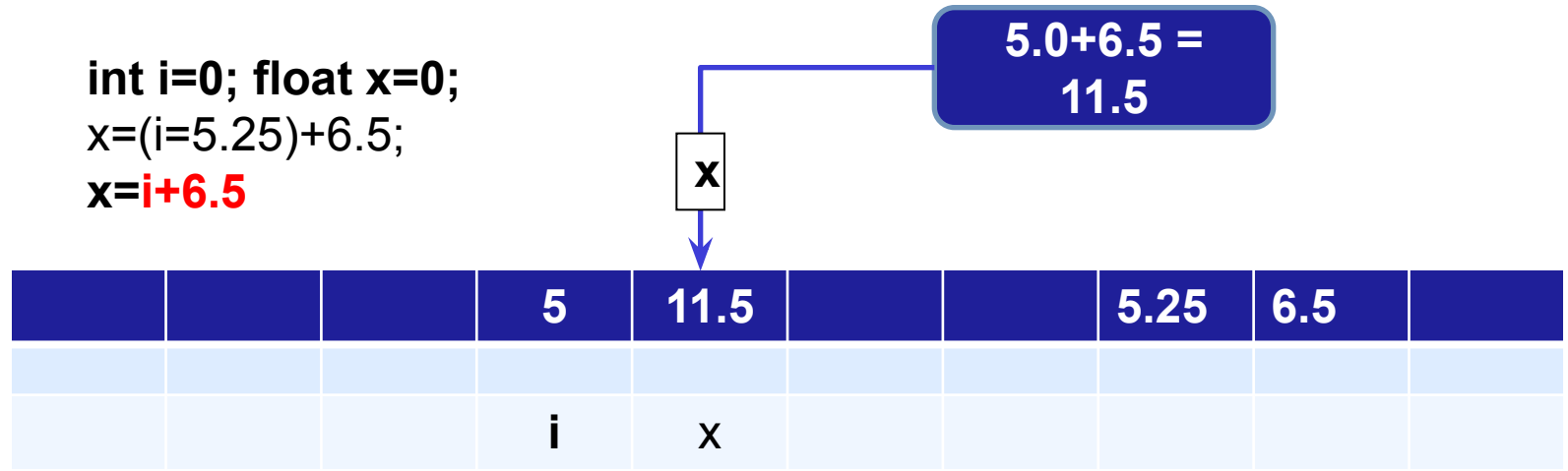
```
int i=0; double x=0;
x=(i=5.25)+6.5;
x=i+6.5
```



Операция присваивания в правой части выражения

- Правильнее: операция присваивания возвращает указатель на переменную (адрес переменной), находящуюся в его левой части.
- Если этот указатель находится в правой части другого оператора присваивания, то он «отдает» значение переменной.

int i=0; float x=0;
x=(i=5.25)+6.5;
x=i+6.5



Множественное присваивание

- В C++ допустимо множественное присваивание:

i=j=k; x=y=z=0;

```
int main()
{
        int i=0; float x=0, y=0; bool b=false;
        x = b = i = y = 4.567;
        cout << y << "  " << i << "  " << b << "  " << x << endl;
//                      4.567          4               1               1
        return 0;
}
```

```
float x=0, y=4.5, z=3.2;
x = (y = z) = 6.5;    // z=3.2   y=6.5     x=6.5
```

Арифметические операции (целые и вещественные операнды)

Оператор	Действие (операнды целые и вещественные, результат - в соответствии с типом операндов)
- +	Присвоение противоположного/сохранение знака
+	Сложение
-	Вычитание
*	Умножение
/	Деление (если применяется к целочисленным операндам – целочисленное деление с отбрасыванием остатка: 5/2 = 2)
%	Деление по модулю (остаток целочисленного деления: 14%3 = 2)
--	Декрементация (уменьшение на 1)
++	Инкрементация (увеличение на 1)

Приоритеты арифметических операций

Оператор	Приоритет
++ --	Высший
унарные + -	
* / %	
*	
+ -	Низший

- Операции, имеющие одинаковый приоритет, выполняются слева направо
- Для изменения порядка выполнения операций применяют скобки

Порядок вычисления выражения, включающего арифметические операции и присваивание

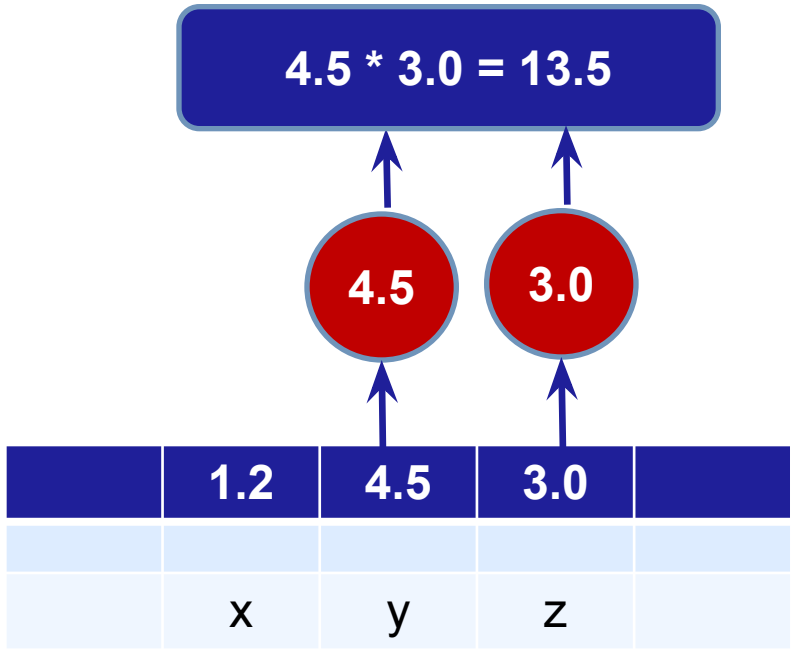
```
double  x=1.2,  y=4.5,  z=3.0;
x = x+y*z;
```

АЛУ

	1.2	4.5	3.0	
	x	y	z	

Порядок вычисления выражения, включающего арифметические операции и присваивание

```
double  x=1.2,  y=4.5,  z=3.0;
x = x+y*z;
```



Порядок вычисления выражения, включающего арифметические операции и присваивание

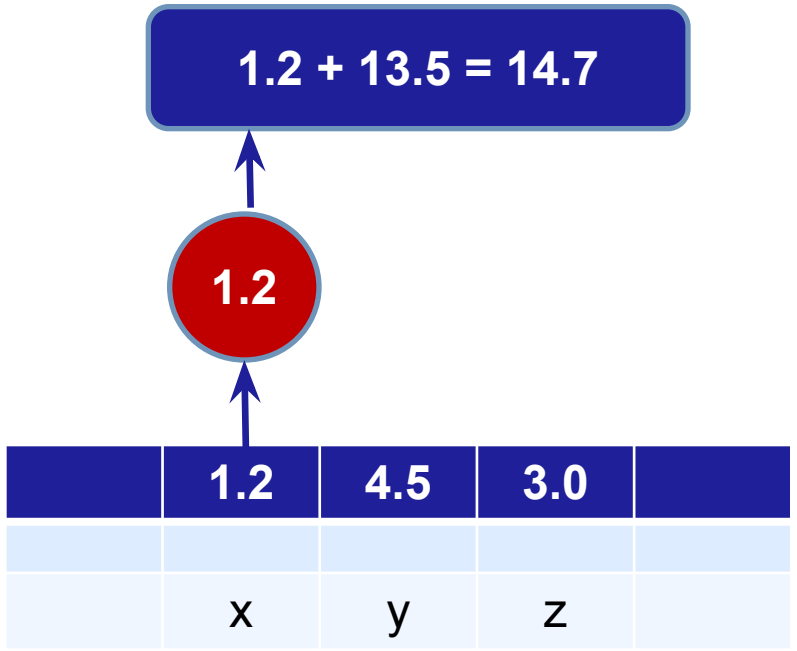
```
double  x=1.2,  y=4.5,  z=3.0;
x = x+y*z;
```

13.5

	1.2	4.5	3.0	
	x	y	z	

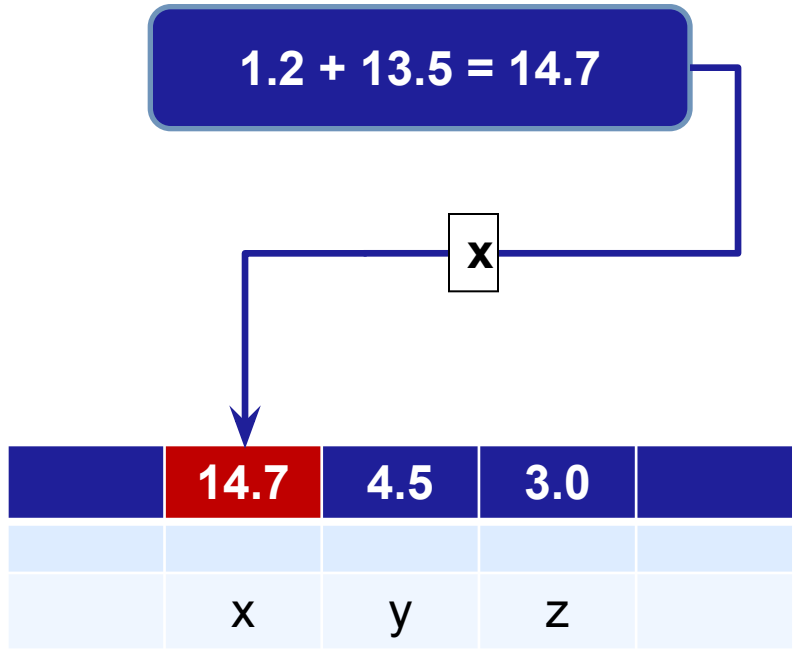
Порядок вычисления выражения, включающего арифметические операции и присваивание

```
double  x=1.2,  y=4.5,  z=3.0;
x = x+y*z;
```



Порядок вычисления выражения, включающего арифметические операции и присваивание

```
double  x=1.2,  y=4.5,  z=3.0;
x = x+y*z;
```



Инкрементация ++ и декрементация --
увеличение и уменьшение значения аргумента на 1

- **Префиксная форма** **++** *Аргумент* **--** *Аргумент*
- **Постфиксная форма** *Аргумент* **++** *Аргумент* **--**

```
int main()
{
    int i=2, j=0;
    i++;          cout << i << endl;                  // 3
    --i;          cout << i << endl;                  // 2
    j=5+i--;      cout << i << "  " << j << endl;          // 1     7
    i=2;
    j=5+ --i;     cout << i << "  " << j << endl;          // 1     6
    return 0;
}
```

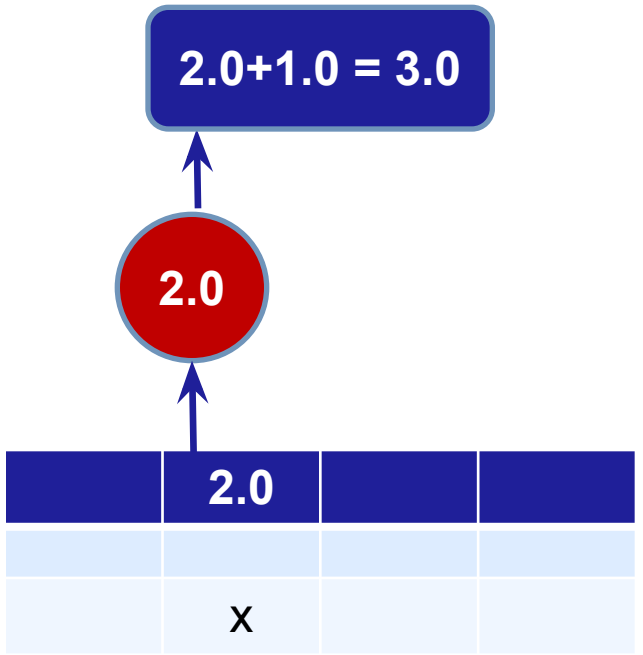
Префиксная и постфиксная формы инкрементации ++ и декрементации –

изменяемая переменная может находиться и в правой, и в левой части операции присваивания:

```
int main()
{
	float x=2;
	x=++x*4; cout << x << endl; // 12
	x=2;
	x=x++*4; cout << x << endl; //  9
	return 0;
}
```

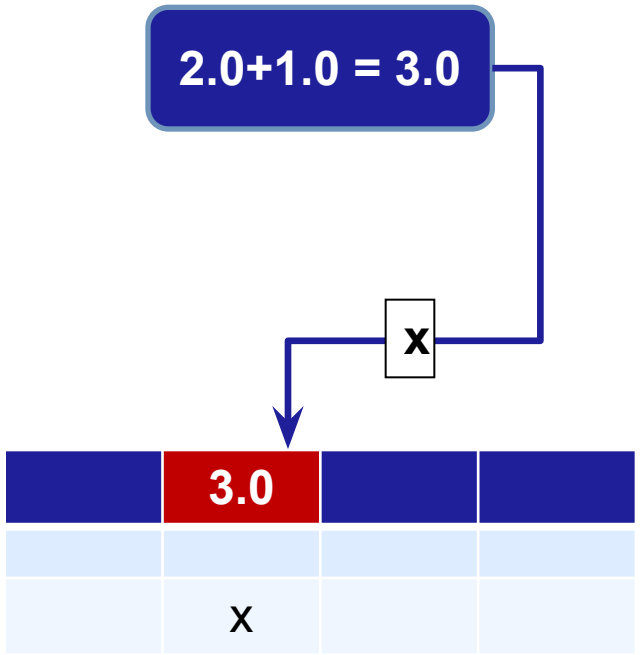
Префиксная и постфиксная формы инкрементации ++ и декрементации --

```
int main()
{
      float x=2;
      x=++x*4; cout << x << endl;      // 12
      x=2;
      x=x++*4; cout << x << endl; //   9
      return 0;
}
```



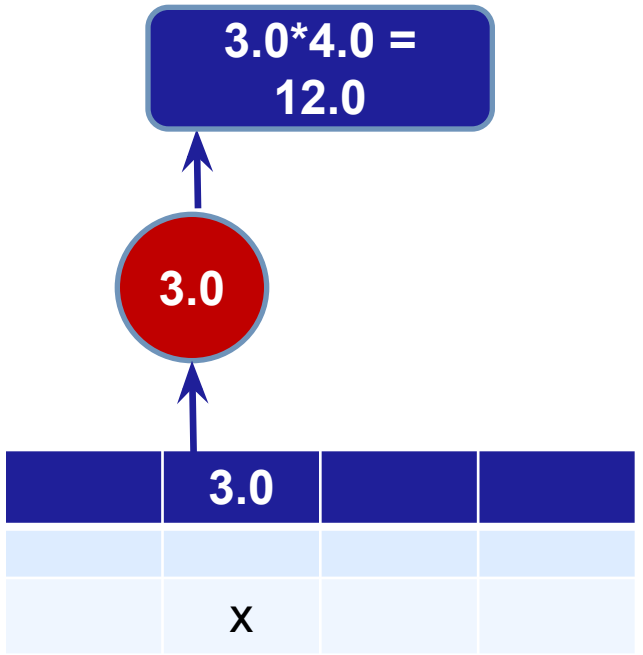
Префиксная и постфиксная формы инкрементации ++ и декрементации --

```
int main()
{
      float x=2;
      x=++x*4; cout << x << endl;      // 12
      x=2;
      x=x++*4; cout << x << endl; //   9
      return 0;
}
```



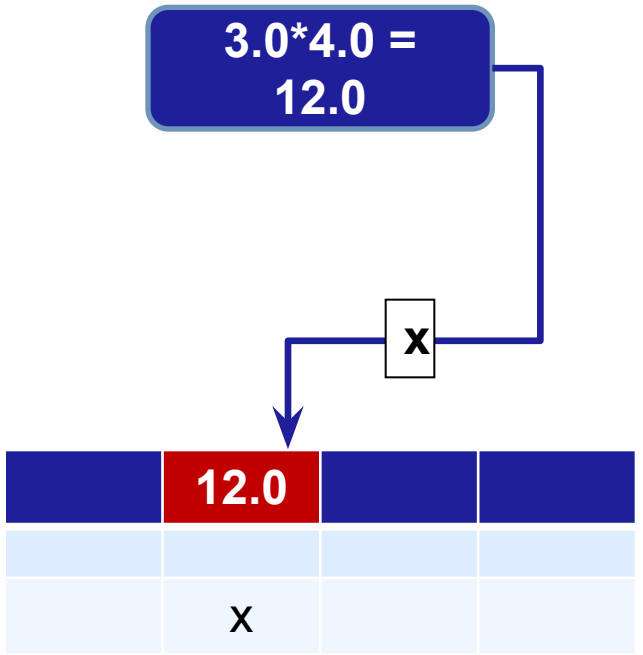
Префиксная и постфиксная формы инкрементации ++ и декрементации --

```
int main()
{
      float x=2;
      x=++x*4; cout << x << endl;      // 12
      x=2;
      x=x++*4; cout << x << endl; //   9
      return 0;
}
```



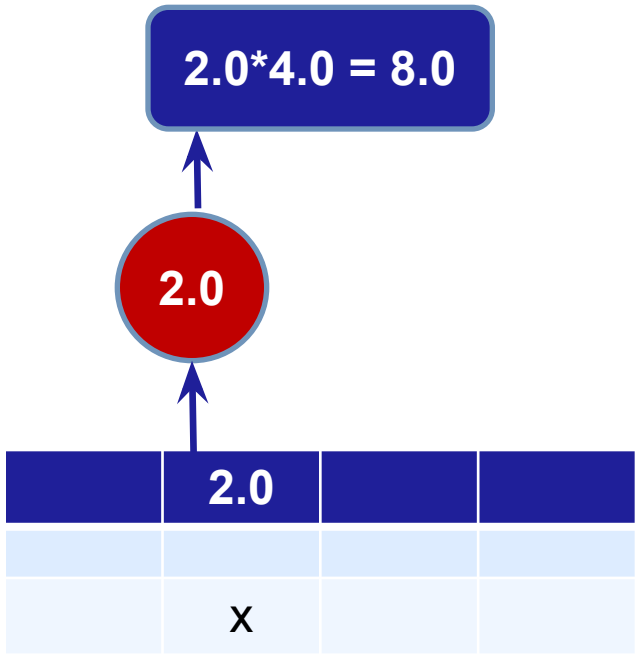
Префиксная и постфиксная формы инкрементации ++ и декрементации --

```
int main()
{
      float x=2;
      x=++x*4; cout << x << endl;     // 12
      x=2;
      x=x++*4; cout << x << endl; //   9
      return 0;
}
```



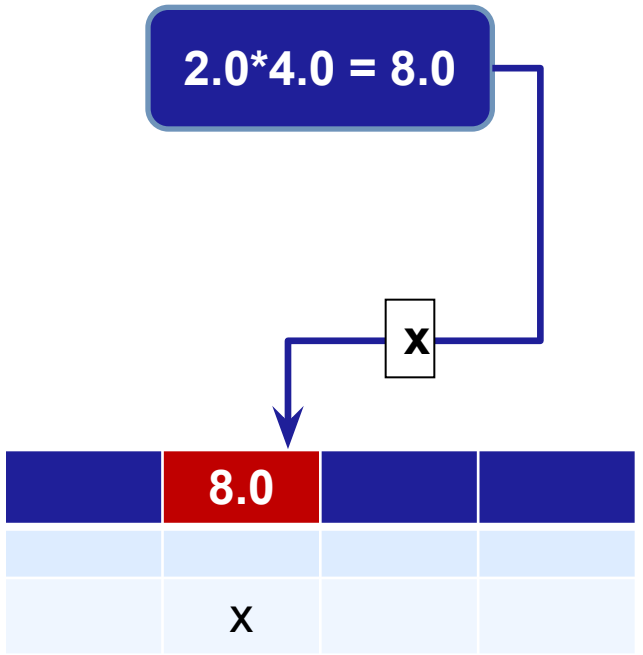
Префиксная и постфиксная формы инкрементации ++ и декрементации --

```
int main()
{
      float x=2;
      x=++x*4; cout << x << endl; // 12
      x=2;
      x=x++*4; cout << x << endl;     //   9
      return 0;
}
```



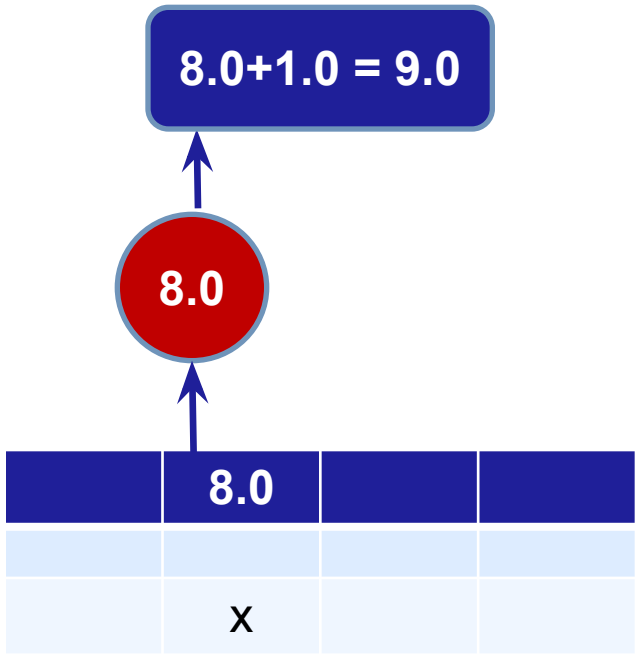
Префиксная и постфиксная формы инкрементации ++ и декрементации --

```
int main()
{
	float x=2;
	x=++x*4; cout << x << endl; // 12
	x=2;
	x=x++*4; cout << x << endl;     //   9
	return 0;
}
```



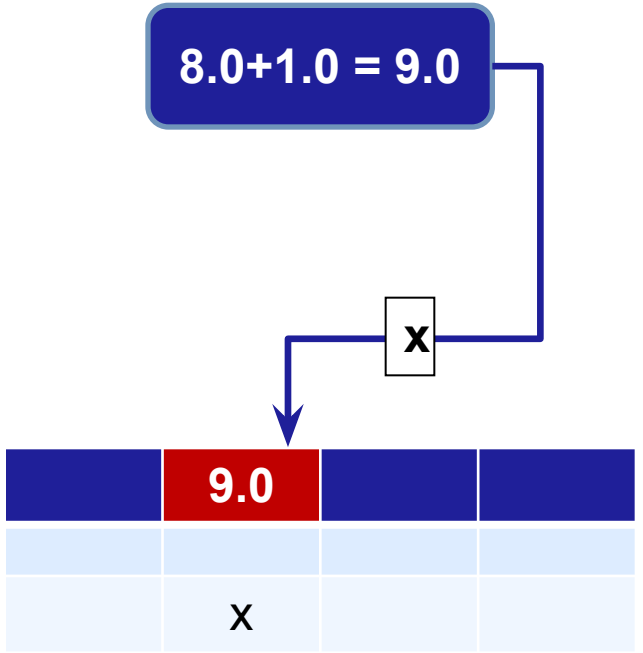
Префиксная и постфиксная формы инкрементации ++ и декрементации --

```
int main()
{
      float x=2;
      x=++x*4; cout << x << endl; // 12
      x=2;
      x=x++*4; cout << x << endl;     //   9
      return 0;
}
```



Префиксная и постфиксная формы инкрементации ++ и декрементации --

```
int main()
{
      float x=2;
      x=++x*4; cout << x << endl; // 12
      x=2;
      x=x++*4; cout << x << endl;     //   9
      return 0;
}
```



Арифметические операции с присваиванием

Оператор	Действие	
+=	Сложение с замещением:	x+=2 эквивалентно x=x+2
-=	Вычитание с замещением:	x-=2 эквивалентно x=x-2
***=**	Умножение с замещением:	x*=2 эквивалентно x=x*2
/=	Деление с замещением:	x/=2 эквивалентно x=x/2
%=	Деление по модулю с замещением:	n%=2 эквивалентно n=n%2

```
double  x=1.2,  y=4.5,  z=3.0;
x += y*z;  эквивалентно x=x+y*z;
```

Операции сравнения и логические операции

Оператор	Операции сравнения. Действие (операнды – целые, вещественные, булевские, указательные, результат – булевский)
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно
!=	Не равно
Оператор	**Логические операции. Действие (операнды – булевские и целые, результат – булевский)**
&&	Логическое И
||	Логическое ИЛИ
!	Логическое НЕ

Операции сравнения

```
int main()
{
	int i=2, j=-3, k=5, l=1;
	char c1='S', c2='U';
	bool b;

	b = i > 3;		cout << b << endl;		// 0
	b = i >=3;		cout << b << endl;		// 0
	b = i < 3;		cout << b << endl;		// 1
	b = k !=2;		cout << b << endl;		// 1
	b = j == i - k;	cout << b << endl;		// 1
	b = 'U' < 'S';		cout << b << endl;		// 0
	b = l == (k > i);	cout << b << endl;		// 1
	return 0;
}
```

Сравнение вещественных чисел на точное равенство

```
int main()
{
    float x=1;  x+=0.5;
    bool b=(x==1.5);  // float x, double1.5  are terminating binary fractions
    cout << x << "   " << b << endl;    // 1.5   1   (True) ☺
    return 0;
}
```

```
int main()
{
   float x=1;  x+=0.1;
   bool b=(x==1.1); // float x, double1.1  are infinite binary fractions
   cout << x << "   " << b << endl;    // 1.1   0   (False)
   return 0;
}
```

Сравнение вещественных чисел на точное равенство

```
int main()
{
    float x=1, z=1.1;  x+=0.1;
    bool b=(x==z); ); // float x, float z are infinite binary fractions
    cout << x       << "   " << b        << endl;   //  1.1                  1   (true) ☺
    cout << x-1.1 << "    " << x-1.1f << endl;   //  2.38419e-008      0   !!!
    return 0;
}
```

```
int main()    // использует  <float.h>
{
      float x=1;    x+=0.1;
      bool b = (abs(x-1.1) <= FLT_EPSILON);       //  OK !!!
      cout << x << "   " << b << endl;              //  1.1       1   (true)
      return 0;
}
```

Логические операции

a	b	a AND b
false	false	false
true	false	false
false	true	false
true	true	true

bool a	bool b	a OR b
false	false	false
true	false	true
false	true	true
true	true	true

bool a	bool b	a XOR b
false	false	false
true	false	true
false	true	true
true	true	false

Не реализована в C и C++

bool a	NOT a
false	true
true	false

0 – false
1 – true

Приоритеты операций сравнения и логических операций

Операторы	Приоритет
!	Высший
> >= < <=	
== !=	
&&	
||	Низший

- Операции сравнения и логические операции имеют более низкий приоритет, чем арифметические операции
- Для изменения порядка выполнения операций применяют скобки

Побитовые операции

Операторы	Действие (операнды – целые, результат – целый)
&	Побитовое И
|	Побитовое ИЛИ
^	Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ
>>	Сдвиг вправо
<<	Сдвиг влево
~	Побитовое НЕ

```
int main()
{
        unsigned short y = 0xFFFF;
        cout << hex << y << endl;
        y = ~y; cout << hex << y << endl;
        return (0);
}
```

y	1111	1111	1111	1111
~y	0000	0000	0000	0000

```
// ffff
// 0
```

Сброс бита четности

```
char  get_char_from_modem ();
{
// читаем из порта модема символ,
// закодированный 7 битами+старший бит четности
      char ch = read_modem ();
      return (ch & 127);      // устанавливаем старший бит в 0
}
```

1 1 0 0 0 0 0 1	ch содержит символ ‘A’ (код 65) +бит четности
0 1 1 1 1 1 1 1	двоичное представление 127
&	побитовый оператор И
0 1 0 0 0 0 0 1	символ ‘A’ без бита четности

Побитовые операции с присваиванием

Знак операции	Действие (операнды – целые, результат – целый)
&=	Побитовое И с замещением
|=	Побитовое ИЛИ м замещением
^=	Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ с замещением
>>=	Сдвиг вправо с замещением
<<=	Сдвиг влево с замещением

```
int main()
{
        unsigned short y = 0x00FF;
        cout << hex << y << endl;  // ff
        y &= 0xAA80;
        cout << hex << y << endl;  // 80
        return (0);
}
```

y	0000	0000	1111	1111
0xAA80	1011	1011	1000	0000
&	0000	0000	1000	0000

Операции побитового сдвига влево и вправо: умножение и деление на степени 2

unsigned char x	Двоичное представление	Десятичное значение
x=7	0 0 0 0 0 1 1 1	7
x = x << 1	0 0 0 0 1 1 1 0	14 = $7*2^1$
x <<= 3	0 1 1 1 0 0 0 0	112 = $14*2^3$
x <<= 2	1 1 0 0 0 0 0 0	192 (потеря разряда)
x = x >> 1	0 1 1 0 0 0 0 0	96 = $192/2^1$
x >>= 2	0 0 0 1 1 0 0 0	24 = $96/2^2$

Операция последовательного вычисления

- Операция последовательного вычисления , (запятая) связывает в одно целое несколько выражений.
- Выражения, разделенные запятой, вычисляются слева направо.
- Если операция «запятая» выполняется в правой части оператора присваивания, то она возвращает значение выражения, находящегося справа (вычисленного последним).
- Например, в результате выполнения:

```
int i=0; float  x=0, y=0;
x= (y=3, y+1);
```
переменная x получит значение 4.

```
x = (y=3.5, i=2, x=y+i, i=x);
```
переменная x получит значение 5.

- Приоритет операции «запятая» ниже, чем операции присваивания.

Другие операции

Формат операции	Название
Выражение_1 **?** *Выражение_2* **:** *Выражение_3*	Операция выбора
&*Идентификатор*	Взятия адреса
***Идентификатор*	Разыменования указателя
sizeof *Идентификатор* **или sizeof (***Тип***)**	Вычисление длины операнда (байт)
*Идентиф_структуры***.***Идентиф_члена*	Доступа к члену структуры
*Указатель_на_структуру***->***Идентиф_члена*	Ссылки на член структуры
Идентификатор **[***Индексное_ выражение***]**	Индексации массива
*Пространство_имен***::***Идентификатор*	Разрешения области действия
(*Выражение***)**	Повышения приоритета операций

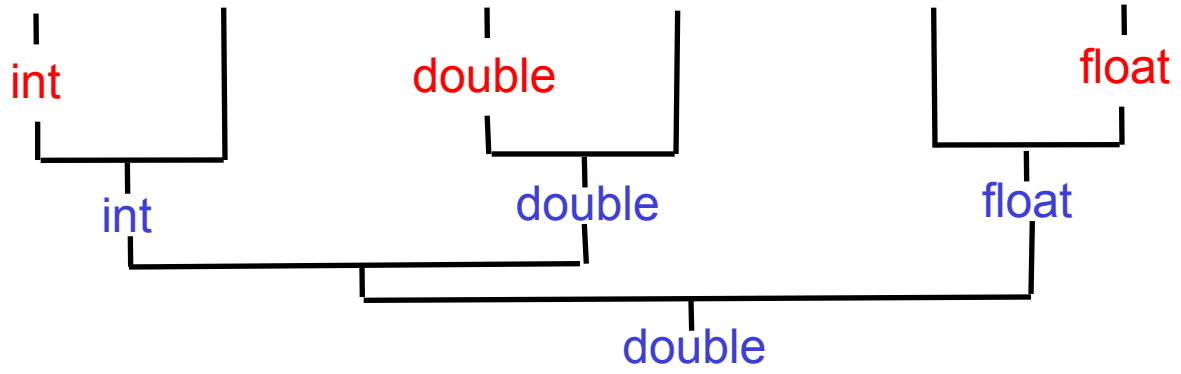
3. Основы программирования
3.21. Приоритет основных операций в C++

Операция	Приоритет
::	**Высший**
. -> [ ] ( )	
++ --	
! ~ ++ -- * (разыменование) & sizeof преобразование типа	
/ % * (умножение)	
+ -	
<< >>	
< <= > >=	
== !=	
|	
&&	
||	
?	
= += -= *= /= %= &= |= ^= >>= <<=	
, (последовательное выполнение операций)	**Низший**

Неявные преобразования типов

- Если в выражение входят переменные и константы различных типов, они, в конечном итоге, преобразуются к одному, «покрывающему», типу.
- Компилятор последовательно преобразовывает операнды каждой операции в тип «покрывающего» операнда, с которым совпадает тип результата.

```
char ch;    int i;     float f;     double d;
result = ( ch  /  i ) + ( f  +  d ) – ( f  +  i )
```



Неявные преобразования типов: иерархия типов данных

Тип данных	старшинство
long double (в MS VS совпадает с double)	**Высший**
double	
float	
unsigned long (в MS VS совпадает с unsigned int)	
long (в MS VS совпадает с int)	
unsigned int	
int	
short (short int)	
unsigned char	
char	
bool	**Низший**

3. Основы программирования
3.22. Преобразование типов в выражениях

Явные преобразования типов

- Для явного приведения типов в C++ можно использовать операцию static_ cast:

Результат = **static_cast <***Требуемый_тип***> (***Аргумент***);**

- Старый синтаксис:

Результат = **(***Требуемый_тип***)** *Аргумент* **;**

или

Результат = *Требуемый_тип* **(***Аргумент***) ;**

- Например:

```
aCharVar = static_cast <char> (anIntVar);
aCharVar = (char) anIntVar;
aCharVar = char (anIntVar);
```

Явные преобразования типов

```
// cast.cpp (от Лафоре)
#include <stdafx.h>
using namespace std;

int main()
  {
   int intVar = 1500000000;                          // 1 500 000 000
   intVar = (intVar * 10) / 10;                      // product  overflow
   cout << "intVar = " << intVar << endl;            //  211509811 ☹

   intVar = 1500000000;
   intVar = (static_cast<double>(intVar) * 10) / 10;      //cast int to double
                                                     // then implicit conversion to int
   cout << "intVar = " << intVar << endl;            // 1 500 000 000  ☺
   return 0;
}
```

Явные преобразования типов (потеря значащих цифр)

```
// cast1.cpp (от Воротницкого)
#include <stdafx.h>
using namespace std;

int main()
   {
   int intVar = 1587654321;                          //1 587 654 321
   intVar = (static_cast<float>(intVar) * 10) / 10;  //cast to float
   cout << "intVar = " << intVar << endl;            // 1 587 654 321 ☺ ???

   intVar = 1587654321;
   float f = static_cast<float>(intVar)              //cast to float
   intVar = (f * 10) / 10;
   cout << "intVar = " << intVar << endl;            // 1 587 654 272  ☹ !!!
   return 0;
   }
```

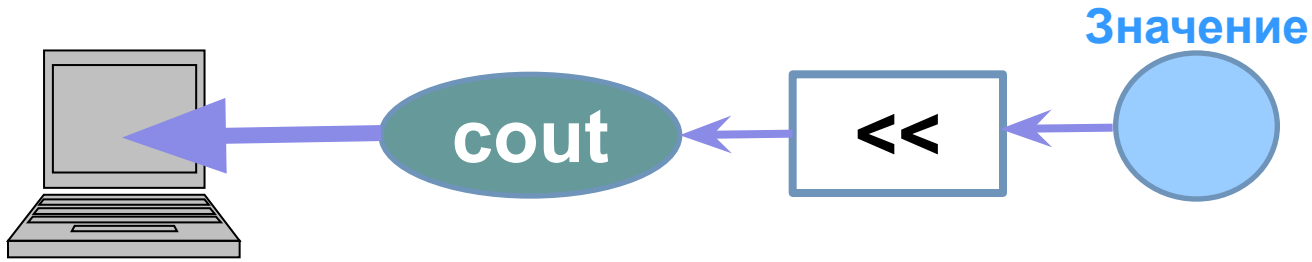
3. Основы программирования
3.23. Стандартные потоки ввода и вывода

```
// Эта программа рассчитывает скорость
// с которой упадет тело, отпущенное с высоты h без начальной скорости
#include <stdafx.h>
using namespace std;
int main()
{
      const float g=9.8;
      float h;
      cout << "Please, enter the value of height (m): ";  cin >>h;
      float v=sqrt(2.0 * g * h);
      cout << "Calculated value of velocity (m/s) is  " << v << endl;
      _getch();
      return 0;
}
```

Объекты для работы с потоками консольного ввода и вывода

Вывод с использованием cout

- Оператор **cout << "Please, enter the value of height (m): ";** выводит на экран значение строковой константы.
- Идентификатор **cout** – объект C++, предназначенный для работы со стандартным потоком вывода.
- Поток – абстрактное понятие, отражающее перемещение данных от приемника к источнику.
- Стандартный поток вывода обычно направлен на экран.
- Операция **<<** называется операцией **вставки**. Она копирует содержание объекта, находящегося в правой части, в объект, содержащийся в левой части.



3. Основы программирования
3.23. Стандартные потоки ввода и вывода

Вывод с использованием cout

- В операторе **cout << "Calculated value of velocity (m/s) is “<< v << endl;** выводятся несколько объектов с использованием каскадирования операции **<<**. Вместо этого пришлось бы писать
  cout << "Calculated value of velocity (m/s) is “;
  cout << v ;
  cout << endl;

- Стандартный поток обеспечивает форматируемый вывод значений различного типа. Поток «знает», как представить на стандартном устройстве целые и вещественные, числа, символы, строки, значения указателей.
- Поэтому нам достаточно записать **cout << v** ; для вывода вещественной переменной.

3. Основы программирования
3.23. Стандартные потоки ввода и вывода

```
// formatted output
// Демонстрирует форматируемый вывод в C++
#include <stdafx.h>
using namespace std;
int main()
      {
      int i=456;     double x=24.6782;    bool b=true;    char c=66;    void* p=&x;
      cout << -125/2 << endl;              // -62
      cout << 5.6e12*2 << endl;            // 1.12e+013
      cout << ! true << endl;              // 0
      cout << i << endl;                   // 456
      cout << x << endl;                   // 24.6782
      cout << b << endl;                   // 1
      cout << c << endl;                   // B
      cout << p << endl;                   // 0013FF50
      _getch(); return 0;  }
```

3. Основы программирования
3.23. Стандартные потоки ввода и вывода

Вывод с использованием cout

- В операторе **cout << "Calculated value of velocity (m/s) is “<< v << endl;** в конце размещен **манипулятор endl.**
- Манипулятор – особая операция, используемая совместно с операцией вставки **<<** , чтобы видоизменять вывод, который делает программа.
- Манипулятор **endl** вставляет в символьный поток символ окончания строки. Весь последующий текст будет печататься с новой строки.
- Практически тот же эффект (строго говоря, endl осуществляет очистку выходного буфера) можно получить вставкой в поток управляющей последовательности \n:
  cout << "Calculated value of velocity (m/s) is “<< v << ‘\n’;
- Еще пример:
  cout << endl; cout << “Rain”; эквивалентно
  cout <<“\nRain”
- .

3. Основы программирования
3.23. Стандартные потоки ввода и вывода

Обозначения управляющих символов в литералах	Символ	Шестнадцатеричный код символа
\a	Сигнал	007
\b	Возврат	008
\f	Перевод страницы	00D
\n	Перевод в начало следующей строки	00A
\r	Возврат каретки	00C
\t	Горизонтальная табуляция	009
\v	Вертикальная табуляция	00B
\\	Обратная косая черта	05C
\’	Одинарные кавычки (апостроф)	022
\”	Двойные кавычки	027
\0	Ноль-символ	000
\DDD	Восьмеричный код символа	
\xDDD	Шестнадцатеричный код символа	

3. Основы программирования
3.23. Стандартные потоки ввода и вывода

- Хотим:
  Please,
  enter the value of height (m):

Способ 1:

```
cout << "Please,\nenter the value of height (m): ";

cout << "Please,"  "\n"  "enter the value of height (m): ";

cout << "Please," ;  cout<<"\n" ;
cout << "enter the value of height (m): ";
```

Способ 2:

```
cout << "Please” << endl << “enter the value of height (m): ";

cout << "Please” ;
cout<< endl;
cout << “enter the value of height (m): “;
```

3. Основы программирования
3.23. Стандартные потоки ввода и вывода

- Хотим:
  Hello, ☼

Способ 1:

‘☼’ соответствует код ASCII $15_{10}=0F_{16}$

```
cout << “Hello, \x0F";
cout << “Hello, ”   “\x0F";
cout << “Hello, ” + “\x0F";
```

Способ 2:

```
cout  << “Hello, ”  << ‘\x0F‘;
cout  << “Hello, ” +‘\x0F‘;
```

Нельзя:

```
cout  << “Hello, ”   ‘\x0F‘;
```

3. Основы программирования
3.23. Стандартные потоки ввода и вывода

```
// width1.cpp
// Демонстрирует необходимость применения манипулятора setw
#include <stdafx.h>
using namespace std;
int main()
    {
    long pop1=2425785, pop2=47, pop3=9761;
    cout << "CITY          " << "POPULATION" << endl
         << "Seattle       " << pop1 << endl
         << "Hightown      " << pop2 << endl
         << "Lowville      " << pop3 << endl;
    _getch();
    return 0;
    }
```

Результат выполнения программы width1.cpp

```
CITY        POPULATION
Seattle      2425785
Hightown    47
Lowville    9761
```

3. Основы программирования
3.23. Стандартные потоки ввода и вывода

```
 // width2.cpp
// Демонстрирует применение манипулятора setw
#include <stdafx.h>
#include <iomanip>
using namespace std;
int main()
    {
    long pop1=2425785, pop2=47, pop3=9761;
    cout << setw(9)<< "CITY" << setw(12)<< "POPULATION" << endl
         << setw(9)<< "Seattle" << setw(12) << pop1 << endl
         << setw(9)<< "Hightown" << setw(12) << pop2 << endl
         << setw(9)<< "Lowville" << setw(12) << pop3 << endl;
    _getch();
    return 0;
    }
```

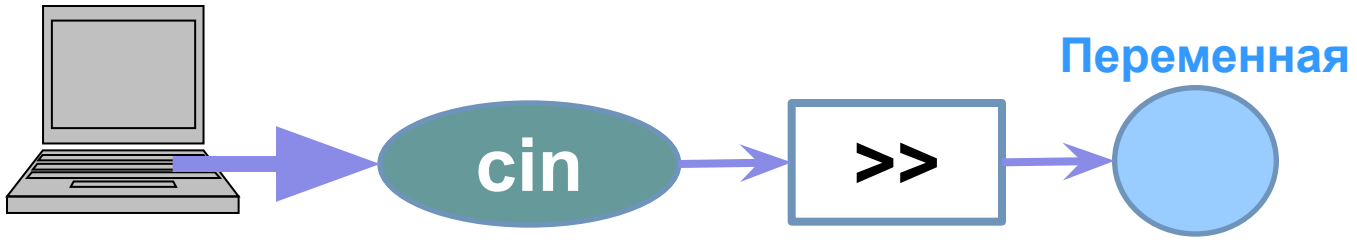
Результат выполнения программы width2.cpp

```
      CITY  POPULATION
   Seattle     2425785
Hightown          47
 Lowville        9761
```

- Манипулятор **setw** устанавливает ширину поля для вывода **следующей за ним** строки или **следующего** числа.
- Выводимая строка или число прижимаются вправо.
- Если параметр, заданный в манипуляторе меньше, чем длина строки или числа, манипулятор игнорируется.

Ввод с использованием cin

- Оператор **cin >>h;** требует ввода из стандартного потока значения, которое будет записано в памяти ЭВМ по адресу, где располагается переменная h.
- Идентификатор **cin** – объект C++, предназначенный для работы со стандартным потоком вывода.
- Стандартный поток ввода обычно содержит данные, вводимые с клавиатуры.
- Операция **>>** называется операцией **извлечения.** Она копирует содержание объекта, находящегося в правой части, в объект, содержащийся в левой части.



Ввод с использованием cin

- Преобразование вводимого значения в двоичное представление производится в соответствии с типом переменной, значение которой вводится (форматированный ввод).
- После ввода значения пользователь должен нажать Enter
- Допускается каскадирование операции **>>** :
  int i; double x; char c;
  cin >> i >> x >> c;
- В этом случае вначале вводится значение i, затем – один или несколько разделителей (пробел или Enter), затем – значение x, затем – разделитель(и), затем – значение c. Завершается ввод нажатием Enter.
- Чтобы ввести символ, не нажимая Enter, следует использовать библиотечные функции _getch() или getche() (с отображением echo), описанные в conio.h. Например, **char c=_getch();**
  Возвращаемое значение можно игнорировать: **_getch();**