Урок 1. Основные элементы языка C/C++. Операции и выражения

Цель работы: изучить алфавит языка C/C++, простые типы данных, научиться объявлять переменные и константы; изучить структуру программы на C/C++.

Основные этапы работы с программой на языке C++ в системе программирования представлены на рис. 1 (прямоугольниками отображены системные программы, а блоки с овальной формой обозначают файлы на входе и на выходе этих программ).

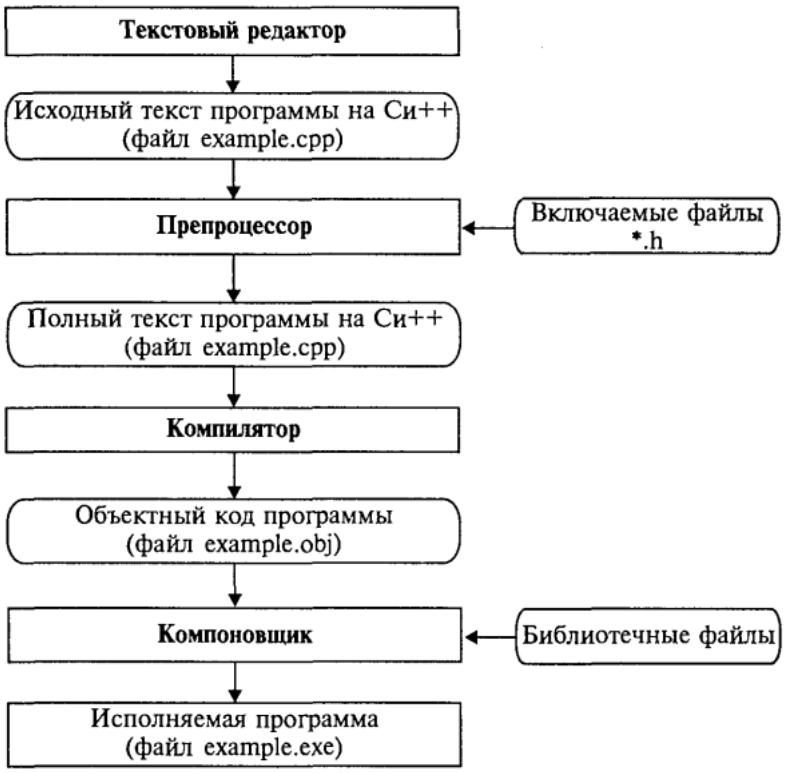

Рис. 1. Этапы работы с программой в системе программирования

1. С помощью текстового редактора формируется текст программы и сохраняется в файле с расширением cpp. Пусть, например, это будет файл с именем example.cpp.
2. Осуществляется этап препроцессорной обработки, содержание которого определяется директивами препроцессора, расположенными перед заголовком программы (функции). В частности, по директиве #include препроцессор подключает к тексту программы заголовочные файлы (*.h) стандартных библиотек.
3. Происходит компиляция текста программы на C++. В ходе компиляции могут быть обнаружены синтаксические ошибки, которые должен исправить программист. В результате успешной компиляции получается объектный код программы в файле с расширением obj. Например, example.obj.
4. Выполняется этап компоновки с помощью системной программы Компоновщик (Linker). Этот этап еще называют редактированием связей. На данном этапе к программе подключаются библиотечные функции. В результате компоновки создается исполняемая программа в файле с расширением exe. Например, example.exe.

Элементы языка C++

Алфавит. В алфавит языка C++ входят:

- латинские буквы: от a до r (строчные) и от A до Z (прописные);
- русские буквы от а до я (строчные) и от А до Я (прописные);
- десятичные цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
- специальные символы: " { } , | [ ] ( ) + - / % \ ; ' : ? < = > _ ! & # ~ ^ . *. К специальным символам относится также пробел.

В комментариях, строках и символьных константах могут использоваться и другие знаки (например, русские буквы).
Комбинации некоторых символов, не разделенных пробелами, интерпретируются как один значимый символ. К ним относятся:

```
++  --  = =  &&  ||  <<  >>  >=  <=  +=  -=  *=  /=  ?:  /*  */  //
```

В C++ в качестве ограничителей **комментариев** могут использоваться как пары символов /* и */, принятые в языке Си, так и символы //, используемые только в C++. Признаком конца такого комментария является невидимый символ перехода на новую строку. Примеры:

```
/* Это комментарий, допустимый в C и C++ */
//Это строчный комментарий, используемый только в C++
```

Из символов алфавита формируются **лексемы** — единицы текста программы, которые при компиляции воспринимаются как единое целое и не могут быть разделены на более мелкие элементы. К лексемам относятся идентификаторы, служебные слова, константы, знаки операций, разделители.

Идентификаторы. Последовательность латинских букв, цифр, символов подчеркивания (_), начинающаяся с буквы или символа подчеркивания, является идентификатором. Например:

```
B12          rus          hard_RAM_disk      MAX      ris_32
```

В C/C++ различаются прописные и строчные буквы. Это значит, что, например, flag, FLAG, Flag, FlAg — разные идентификаторы.

Ограничения на длину идентификатора могут различаться в разных реализациях языка. Стандарт ANSI языка C/C++ позволяет использовать до 31 первых символов имени.

Служебные (ключевые) слова. Служебные слова в C — это идентификаторы, назначение которых однозначно определено в языке. Они не могут быть использованы как свободно выбираемые имена. Полный список служебных слов зависит от реализации языка, т. е. различается для разных компиляторов. Однако существует неизменное ядро, которое определено стандартом C++. Вот его список:

```
asm       auto      break     case      catch     char      class     const
continue  default   delete    do        double    else      enum      extern
float     for       friend    goto      if        inline    int       long
new       operator  private   protected  public    register  return    short
signed    sizeof    static    struct    switch    template  this      throw
```

```
try      typedef  typeid   union     unsigned virtual  void
volatile while
```

Дополнительные к этому списку служебные слова приведены в описаниях конкретных реализаций C++. Некоторые из них начинаются с символа подчеркивания, например: _export, _ds, _AH и др. Существуют служебные слова, начинающиеся с двойного подчеркивания. В связи с этим программисту не рекомендуется использовать в своей программе идентификаторы, начинающиеся с одного или двух подчеркиваний, во избежание совпадения со служебными словами.

Типы данных

Концепция **типов данных** является важнейшей стороной любого языка программирования. Особенность Схема типов для языка C++ представлена на рис. 2.

Рис. 2. Типы данных, используемых в программах на языке C/C++

В C/C++ имеется четыре базовых арифметических (числовых) типа данных. Из них два целочисленных — char, int — и два плавающих (вещественных) — float и double. Кроме того, в программах можно использовать некоторые модификации этих типов, описываемых с помощью служебных слов — модификаторов.

Существуют два **модификатора размера** — short (короткий) и long (длинный) — и два модификатора знаков — signed (знаковый) и unsigned (беззнаковый). Знаковые модификаторы применяются только к целым типам.

Как известно, тип величины связан с ее формой внутреннего представления, множеством принимаемых значений и множеством операций, применимых к этой величине.

В табл. 1 перечислены **арифметические** типы данных C++, указан объем занимаемой памяти и диапазон допустимых значений.

Размер типа int и unsigned int зависит от размера слова операционной системы, в которой работает компилятор C++. Например, в 16-разрядных ОС (MS DOS) этим типам соответствуют 2 байта, в 32-разрядных (Windows) — 4 байта.

Таблица 1. Арифметические типы данных

Тип данных	**Размер (байт)**	**Диапазон значений**	**Эквивалентные названия типа**
char	1	-128...+ 127	signed char
int	2/4	зависит от системы	signed, signed int
unsigned char	1	0...255	нет
unsigned int	2/4	зависит от системы	unsigned
short int	2	-32768... 32767	short, signed short int
unsigned short	2	0...65535	unsigned short int
long int	4	-2147483648... 2147483647	long, signed long int
unsigned long int	4	0...4294967295	unsigned long
float	4	±(3.4E-38...3.4E+38)	нет
double	8	±(1.7E-308... 1.7E+308)	нет
long double	10	±(3.4E-4932...1.1E+4932)	нет

Анализируя данные табл. 1, можно сделать следующие выводы:

- если не указан базовый тип, то по умолчанию подразумевается int;
- если не указан модификатор знаков, то по умолчанию подразумевается signed;
- с базовым типом float модификаторы не употребляются;
- модификатор short применим только к базовому типу int.

В C/C++ величины типа char могут рассматриваться в программе и как **символы**, и как **целые числа**. Все зависит от контекста, т.е. от способа использования этой величины. В случае интерпретации величины типа char как символа ее числовое значение является ASCII-кодом. Следующий пример иллюстрирует сказанное.

```
char  a = 65;
printf("%c", a);  /*На  экране появится символ A*/
printf("%d", a);  /*На  экране появится число 65*/
```

Символы "%c" являются спецификацией формата ввода/вывода символьных данных, а "%d" — спецификацией для целых чисел.

В качестве **логических** величин в C/C++ выступают целые числа. Интерпретация их значений в логические величины происходит по правилу: равно нулю — ложь (false), не равно нулю — истина (true). В последние версии C++ добавлен отдельный логический тип с именем bool. Его относят к разновидности целых типов данных.

Описание **переменных** в программах на C/C++ имеет вид:

```
имя_типа список_переменных;
```

Примеры описаний:

```
char  symbol, cc;
unsigned char code;
int number, row;
unsigned long long_number;
```

```
float x, X, cc3;
double e, b4;
long double max_num;
```

Одновременно с описанием можно задать начальные значения переменных. Такое действие называется **инициализацией** переменных. Описание с инициализацией производится по следующей схеме:

```
тип  имя_переменной = начальное_значение
```

Например:

```
float  pi = 3.14159,  c = 1.23;
unsigned  int  year = 2000;
```

Константы. Запись целых констант. ***Целые десятичные*** числа, начинающиеся не с нуля, например: 4, 356, -128.

Целые восьмеричные числа, запись которых начинается с нуля, например: 016, 077.

Целые шестнадцатеричные числа, запись которых начинается с символов 0x, например:

```
0x1A,  0x253, 0xFFFF.
```

Тип константы компилятор определяет по следующим правилам: если значение константы лежит в диапазоне типа int , то она получает тип int; в противном случае проверяется, лежит ли константа в диапазоне типа unsigned int , в случае положительного ответа она получает этот тип; если не подходит и он, то пробуется тип long и, наконец, unsigned long.
Если значение числа не укладывается в диапазон типа unsigned long, то возникает ошибка компиляции.

Запись вещественных констант. Если в записи числовой константы присутствует десятичная точка (2.5) или экспоненциальное расширение (1E-8), то компилятор рассматривает ее как вещественное число и ставит ей в соответствие тип double. Примеры вещественных констант:

```
44.  3.14159  44E0     1.5E-4.
```

Использование суффиксов. Программист может явно задать тип константы, используя для этого суффиксы. Существуют три вида суффиксов: F(f) - float; U(u) - unsigned; L(l) - long (для целых и вещественных констант). Кроме того, допускается совместное использование суффиксов и L в вариантах UL или LU.

Примеры:

```
3.14159F  — константа типа float, под которую выделяется 4 байта памяти;
3.14L  — константа  типа  long  double,  занимает  10 байт;
50000U  — константа  типа  unsigned  int,  занимает  2  байта памяти
          (вместо четырех без суффикса);
0LU  — константа  типа  unsigned  long,  занимает  4 байта;
24242424UL  — константа  типа  unsigned  long,  занимает  4 байта.
```

Запись символьных и строковых констант. Символьные константы заключаются в апострофы. Например: 'A', 'a', '5', ' + '. Строковые константы, представляющие собой

символьные последовательности, заключаются в двойные кавычки. Например: "rezult", "введите исходные данные".

Особую разновидность символьных констант представляют так называемые **управляющие символы.** Их назначение — управление выводом на экран. Как известно, такие символы расположены в начальной части кодовой таблицы ASCII (коды от 0 до 31) и не имеют графического представления. В программе на С они изображаются парой символов, первый из которых '\'. Вот некоторые из управляющих символов:

```
' \n '  — переход на новую строку;
' \t '  — горизонтальная  табуляция;
' \a '  —  подача  звукового  сигнала.  Полный  список  управляющих сим-
          вольных  последовательностей  будет дан  позднее.
```

Управляющие символьные последовательности являются частным случаем эскейп-последовательностей (ESC-sequence), с помощью которых можно задать символьную константу указанием ее кода. Код символа можно указать в восьмеричном или в шестнадцатеричном представлении. Формат восьмеричного представления:' \ddd'. Здесь d — восьмеричная цифра (от 0 до 7). Формат шестнадцатеричного представления:' \xhh' (или ' \xhh'), где h — шестнадцатеричная цифра (от 0 до F). Например, константа, соответствующая заглавной латинской букве А, может быть представлена тремя способами:

```
'A',        '\101',    '\x41'.
```

Именованные константы (константные переменные). В программе на С/С++ могут использоваться именованные константы. Употребляемое для их определения служебное слово const принято называть **квалификатором** доступа. Квалификатор const указывает на то, что данная величина не может изменяться в течение всего времени работы программы. В частности, она не может располагаться в левой части оператора присваивания. Примеры описания константных переменных:

```
const float pi = 3.14159;
const int iMIN = 1,  iMAX = 1000;
```

Определение констант на стадии препроцессорной обработки программы. Еще одной возможностью ввести именованную константу является использование препроцессорной директивы #define в следующем формате:

```
#define <имя константы> <значение константы>
```

Например:

```
#define iMIN 1
#define iMAX 1000
```

Тип констант явно не указывается и определяется по форме записи. В конце директивы точка с запятой не ставится.

На стадии препроцессорной обработки указанные имена заменяются на соответствующие значения. Например, если в программе присутствует оператор

```
X = iMAX - iMIN;
```

то в результате препроцессорной обработки он примет вид:

```
X = 1000 - 1;
```

При этом идентификаторы iMAX и iMIN не требуют описания внутри программы.

Константы перечисляемого типа. Данное средство языка позволяет определять последовательность целочисленных именованных констант. Описание перечисляемого типа начинается со служебного слова enum, а последующий список констант заключается в фигурные скобки. Например:

```
enum  {A, B, C, D};
```

В результате имени A будет сопоставлена константа 0, имени B — константа 1, C — 2, D — 3. По умолчанию значение первой константы равно нулю. Для любой константы можно явно указать значение. Например:

```
enum  {A = 10, B, C, D};
```

В результате будут установлены следующие соответствия:

```
A = 10,  B = 11,  C = 12,  D = 13.
```

Возможен и такой вариант определения перечисления:

```
enum  {A = 10, B = 20, C = 35, D = 100};
```

Если перечисляемому типу дать имя, то его можно использовать в описании переменных. Например:

```
enum metal {Fe, Co, Na, Cu, Zn};
metal  Met1, Met2;
```

Здесь идентификатор metal становится именем типа. После такого описания в программе возможны следующие операторы:

```
Met1 = Na;  Met2 = Zn;
```

Операции и выражения

Во всех языках программирования под **выражением** подразумевается конструкция, составленная из констант, переменных, знаков операций, функций, скобок. Выражение определяет порядок вычисления некоторого значения. Если это числовое значение, то такое выражение называют арифметическим. Вот несколько примеров арифметических выражений, записанных по правилам языка C:

```
a + b     12.5 - z          2 * (X + Y)
x++        x++ b            --n * 2        n* = l
```

Три первых выражения имеют традиционную форму для языков программирования высокого уровня, поэтому их смысл очевиден. Следующие четыре выражения специфичны для языка C.

Набор операций, используемых в C, а также правила записи и вычисления выражений. Операция, применяемая к одному операнду, называется унарной, а операция с двумя операндами — бинарной.

Арифметические операции. К арифметическим операциям относятся:

- вычитание или унарный минус;
+ сложение или унарный плюс;
* умножение;
/ деление;

% деление по модулю;

++ унарная операция увеличения на единицу (инкремент);

-- унарная операция уменьшения на единицу (декремент).

Все операции, кроме деления по модулю, применимы к любым числовым типам данных. Операция % применима только к целым числам.

Особенности выполнения операции деления: если делимое и делитель — целые числа, то и результат — целое число. Например, значение выражения 5/3 будет равно 2, а при вычислении 1/5 получится 0.

Если хотя бы один из операндов имеет вещественный тип, то и результат будет вещественным. Например, операции 5./3, 5./3., 5/3. дадут вещественный результат 1.6666.

Операции инкремента и декремента могут применяться только к переменным и не могут — к константам и выражениям. Операция ++ увеличивает значение переменной на единицу, операция — уменьшает значение переменной на единицу. Оба знака операции могут записываться как перед операндом (префиксная форма), так и после операнда (постфиксная форма), например: ++x или x++, --a или a--. Три следующих оператора дают один и тот же результат:

```
x = x + 1;     ++x;      x++
```

Различие проявляется при использовании префиксной и постфиксной форм в выражениях. Проиллюстрируем это на примерах.

Первый пример:

```
a = 3;  b = 2;
c = a++ * b++;
```

В результате выполнения переменные получат следующие значения: a = 4, b = 3, c = 6.

Второй пример:

```
a = 3;  b = 2;
c = ++a * ++b;
```

Результаты будут такими: a = 4, b = 3, c = 12.

Объяснение следующее: при использовании постфиксной формы операции ++ и - - выполняются после того, как значение переменной было использовано в выражении, а префиксные операции — до использования. Поэтому в первом примере значение переменной с вычислялось как произведение 3 на 2, а во втором — как произведение 4 на 3.

По убыванию старшинства арифметические операции расположены в следующем порядке:

```
++,  - -
-  (унарный минус)
*,  ^  %
+,   -
```

Одинаковые по старшинству операции выполняются в порядке слева направо. Для изменения порядка выполнения операций в выражениях могут применяться круглые скобки.

Операции отношения.

< меньше,

<= меньше или равно,

> больше,
>= больше или равно,
= равно,
!= не равно.

Результатом операции отношения является целое число: если отношение истинно — то 1, если ложно — то 0.

Примеры отношений:

```
a < 0,  101 >= 105,  'a' == 'A'   'a' != 'A'
```

Результатом второго и третьего отношений будет 0 — ложь; результат четвертого отношения равен 1 — истина; результат первого отношения зависит от значения переменной a.

Логические операции.

! операция отрицания (НЕ),
&& конъюнкция, логическое умножение (И),
|| дизъюнкция, логическое сложение (ИЛИ).

Правила их выполнения определяются таблицей истинности (см. табл. 2).

Например, логическое выражение, соответствующее системе неравенств $0 < x < 1$ в программе на С запишется в виде следующего логического выражения:

```
x  > 0  &&  x < 1
```

В С операции отношения старше конъюнкции и дизъюнкции. По убыванию приоритета логические операции и операции отношения расположены в следующем порядке:

> < >= <=
&&
||

Помимо рассмотренных в С имеются **поразрядные (битовые) логические операции**. Эти операции выполняются над каждой парой соответствующих двоичных разрядов внутреннего представления операндов. Их еще называют битовыми логическими операциями. Знаки битовых логических операций:

& поразрядная конъюнкция (И),
| поразрядная дизъюнкция (ИЛИ),
^ поразрядное исключающее ИЛИ,
~ поразрядное отрицание (НЕ).

Битовые логические операции вместе с операциями поразрядного сдвига влево (<<) и вправо (>>) позволяют добраться до каждого бита внутреннего кода. Чаще всего такие действия приходится выполнять в системных программах.

Операция присваивания. Знак операции присваивания =. Следствием отмеченного факта является то, что присваивание, как любой другой знак операции, может несколько раз входить в выражение. Например:

```
a = 10 = c = x + y;
```

Присваивание имеет самый низкий приоритет (ниже только у операции «запятая»). Кроме того, операция присваивания — правоассоциативная. Это значит, что несколько подряд расположенных присваиваний выполняются справа налево. Поэтому в приведенном выше выражении первой выполнится операция сложения, затем переменной с присвоится значение суммы, затем это значение присвоится переменной /> и в конце — переменной a.

В языке C имеются дополнительные операции присваивания, совмещающие присваивание с выполнением других операций. Среди них: +=, -=, /=, *=, %=. Приоритет у них такой же, как и у простого присваивания. Примеры использования этих операций:

```
a += 2  эквивалентно  a = a + 2,
x -= a + b  эквивалентно  x = x - (a + b),
p /= 10  эквивалентно  p = p/10,
m *= n  эквивалентно  m = m * n,
r %= 5  эквивалентно  r = r % 5.
```

Заметим, что вместо выражения a = a + 2 предпочтительнее писать в программе a+=2, поскольку второе выражение будет вычисляться быстрее.

Операция явного преобразования типа (операция «тип»). Применение этой операции имеет следующий формат:

```
(имя_типа)  операнд
```

Операндом могут быть константа, переменная, выражение. В результате значение операнда преобразуется к указанному типу. Примеры использования преобразования типа:

```
(long)8,  (float)1 ,  (int)x%2
```

По поводу последнего выражения заметим, что приоритет операции «тип» выше деления (и других бинарных арифметических операций), поэтому сначала значение переменной x приведется к целому типу (отбросится дробная часть), а затем выполнится деление по модулю.

Следующий фрагмент программы иллюстрирует одну из практических ситуаций, в которой потребовалось использовать преобразование типа:

```
float  c;
int  a = 1,  b = 2;
c = ( float ) a / b ;
```

В результате переменная c получит значение 0,5. Без преобразования типа ее значение стало бы равно 0.

Операция sizeof. Эта операция имеет две формы записи:

```
sizeof(тип)  и  sizeof(выражение)
```

Результатом операции является целое число, равное количеству байтов, которое занимает в памяти величина явно указанного типа или величина, полученная в результате вычисления выражения. Последняя определяется также по типу результата выражения. Хотя по форме записи это похоже на функцию, однако sizeof является именно операцией. Ее приоритет выше, чем у бинарных арифметических операций, логических операций и отношений. Примеры использования операции:

```
sizeof ( int )     результат — 2
sizeof  (1)        результат — 2
sizeof  (0.1)      результат — 8
sizeof (1L)        результат — 4
sizeof (char) результат — 1
sizeof ( 'a' )результат — 2
```

Операция «запятая». Эта необычная операция используется для связывания нескольких выражений в одно. Несколько выражений, разделенных запятыми, вычисляются последовательно слева направо. В качестве результата такого совмещенного выражения принимается значение самого правого выражения. Например, если переменная x имеет тип int, то значение выражения (x = 3, 5*x) будет равно 15, а переменная x примет значение 3.

Операция «условие ?:». Это единственная операция, которая имеет три операнда. Формат операции:

```
выражение1 ? выражение2 : выражение3
```

Данная операция реализует алгоритмическую структуру ветвления. Алгоритм ее выполнения следующий: первым вычисляется значение выражения 1, которое обычно представляет собой некоторое условие. Если оно истинно, т. е. не равно 0, то вычисляется выражение 2 и полученный результат становится результатом операции. В противном случае в качестве результата берется значение выражения 3.

Пример 1. Вычисление абсолютной величины переменной X можно организовать с помощью одной операции:

```
X < 0  ?  -X  :  X;
```

Пример 2. Выбор большего значения из двух переменных a и b:

```
max = (a <= b) ? b:a;
```

Пример 3. Заменить большее значение из двух переменных a и b на единицу:

```
( a > b ) ? a : b = 1 ;
```

Правила языка в данном случае позволяют ставить условную операцию слева от знака присваивания.

Операции ( ) и [ ]. В языке C круглые и квадратные скобки рассматриваются как операции, причем эти операции имеют наивысший приоритет.

Подведем итог всему разговору об операциях C/C++, сведя их в общую табл. 2 и расположив по рангам. Ранг операции — это порядковый номер в ряду приоритетов. Чем больше ранг, тем ниже приоритет. В таблице отражено еще одно свойство операций — ассоциативность. Если одна и та же операция, повторяющаяся в выражении несколько раз, выполняется в порядке расположения слева направо, то она называется левоассоциативной; если выполняется справа налево, то операция правоассоциативная. В таблице эти свойства отображены стрелками влево и вправо. Некоторые операции, присутствующие в таблице, пока не обсуждались.

Таблица 2. Приоритеты (ранги) операций

Ранг	Операции	Ассоциативность
1	( ) [ ] - > .	→
2	! ~ + - ++ -- & * (тип) sizeof (унарные)	←
3	* / % (мультипликативные бинарные)	→
4	+ - (аддитивные бинарные)	→
5	<< >>> (поразрядного сдвига)	→
6	< <= >= > (отношения)	→
7	= = != (отношения)	→
8	& (поразрядная конъюнкция «И»)	→
9	^ (поразрядное исключающее «ИЛИ»)	→
10	| (поразрядная дизъюнкция «ИЛИ»)	→
11	&& (конъюнкция «И»)	→
12	|| (дизъюнкция «ИЛИ»)	→
13	?: (условная)	←
14	= *= /= %= += -= &= ^= |= << = >>=	←
15	, («запятая»)	→

Приведение типов при вычислении выражений. Практически во всех языках программирования высокого уровня работает ряд общих правил записи выражений:

- все символы, составляющие выражение, записываются в строку (нет надстрочных и подстрочных символов);
- в выражении проставляются все знаки операций;
- при записи выражения учитываются приоритеты операций;
- для влияния на последовательность операций используются круглые скобки.

Некоторые специфические особенности записи выражений на С были описаны выше при рассмотрении операций языка.

В процессе вычисления выражений с разнотипными операндами производится автоматическое преобразование типов величин.

Знание программистом правил, по которым происходят эти преобразования, предупреждает некоторые ошибки в записи выражений. Суть правил преобразования при выполнении бинарных операций сводится к следующему:

- преобразование не выполняется, если оба операнда имеют одинаковый тип;
- при разных типах операндов происходит приведение величины с младшим типом к старшему типу (кроме операции присваивания);
- при выполнении операции присваивания величина, полученная в правой части, преобразуется к типу переменной, стоящей слева от знака =.

Старшинство типов друг по отношению к другу определяется по следующему принципу: старший тип включает в себя все значения младшего типа как подмножество. Вещественные (плавающие) типы являются старшими по отношению к целым. В свою очередь, внутри каждой из этих групп иерархия типов устанавливается в соответствии с указанным принципом. Целые типы по возрастанию старшинства расположены в таком порядке:

```
char -> shot -> int -> long
```

Порядок старшинства вещественных типов следующий:

```
float -> double -> long  double
```

Следует иметь в виду, что при преобразовании целой величины к плавающему типу может произойти потеря точности (вместо 1 получится 0,999).

Следующий пример иллюстрирует порядок выполнения операций и происходящие преобразования типов при вычислении выражения (рис. 3, цифры вверху — порядок операций).

```
char ch; int i; float f;
double d; long double r;
```

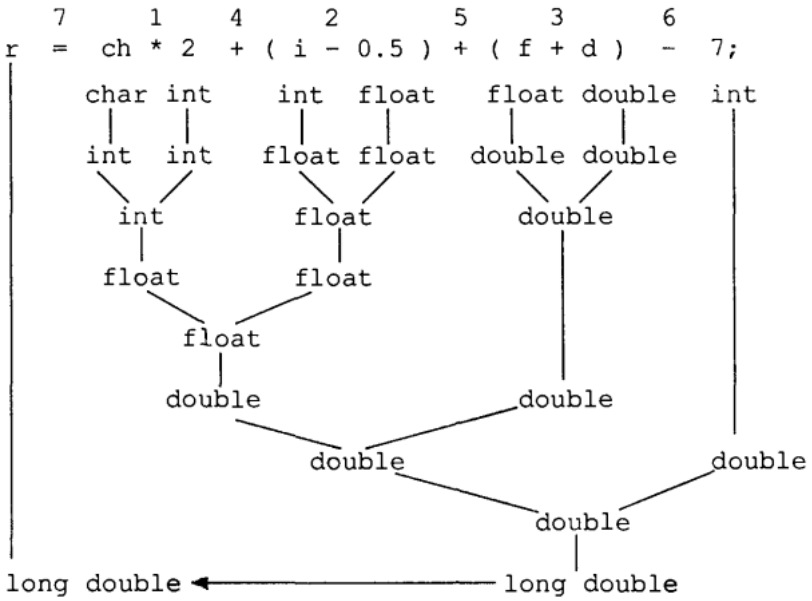

Рис. 3. Порядок выполнения операций и происходящие преобразования типов

Упражнения

1. Определить тип константы:

```
а)  315;      б)  -32.4;    в)  102408;   г)  3.7E57;        д)  0315;
е)  0x24;     ж)  2.6L;     з)  70700U;   и)  ' 5 ' ;        к)  ' \121'
```

2. В программе объявлена переменная: int n = 10. Определить результаты вычислений следующих выражений:

```
а) n++;         б)++n;          в) n%2;          г) n/3;     д) n/3.;
е) ++n + 5;     ж) 5+n++;       з) (float)n/4;
и) sizeof(n);   к) sizeof(1.*n).
```

3. Координаты точки на плоскости заданы переменными X и Y.

Записать следующие условия в форме логических выражений:

а) точка лежит в первой четверти координатной плоскости;
б) точка лежит на оси X;
в) точка лежит на одной из осей;
г) точка лежит в первой или второй четверти внутри единичной окружности;
д) точка лежит на единичной окружности в третьей или четвертой четверти;
е) точка лежит внутри кольца с внутренним радиусом 1 и внешним радиусом 2 во второй или четвертой четверти.

4. В программе объявлена переменная: float x = 2. Какое значение получит переменная x в результате вычисления следующих выражений?

```
а)  x += 2;  б)  x /= 10;  в)  x *= (x + 1);  г) x += x += x += 1.
```

5. Определить значения выражений для трех вариантов объявления переменной x:

```
1) float  x = 1.;       а) x > 1 ? 2*x : x;
2) float  x = 10.;      б) x/5 == 2 ? 5 : x/10;
3) int  x = 1.          в) x > 0 && x <= 1 ? 1:0
```

Литература

1. **Семакин И.Г., Шестаков А.П.** Основы программирования: Учебник.- М.: Мастерство, 2002.- 432 с.