# ЛЕКЦИЯ №2

## ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

*Описан материал, который имеет отношение к основам программирования на алгоритмическом языке СИ (алфавит, операции, операторы, комментарии и другие элементы). Приводятся примеры программ, иллюстрирующие применение основных операторов СИ.*

### Основные понятия языка СИ

Программы на СИ представляют собой последовательность строк, которые в отдельности состоят из символов.

Набор символов включает в себя 26 букв латинского алфавита или (и) 33 буквы русского алфавита, десять арабских цифр от нуля до девяти и некоторое конечное множество специальных символов, например: ;, :, !, =, и т. д. Со всем набором символов познакомимся по мере изучения языка СИ.

Таким образом, текст программы представляет собой информацию о последовательности действий, которые должна выполнить ЭВМ при решении той задачи, алгоритм решения которой она представляет. Причем данная информация закодирована в виде набора символов.

Основные понятия необходимые для написания программы на алгоритмическом языке СИ будут вводиться по мере необходимости. Программа работает с такими основными объектами, как переменные и константы.

2

*Константами* являются такие объекты программы, которые за все время ее выполнения ни разу не изменяли своего значения, а *переменные* – объекты, которые во время выполнения программы могут изменять свое значение сколь угодно раз. Таких объектов в программе может быть много и их необходимо различать друг от друга. Для этой цели служат *идентификаторы*.

*Идентификаторы*. Под ними понимается буквенно-цифровая последовательность, которая начинается с буквы и не должна совпадать с каким-либо ключевым словом. Количество символов в идентификаторе (имени) может быть сколь угодно. Однако значащими являются восемь первых символов или, например, 32 в Turbo C. В именах различаются строчные и прописные буквы. Символ _ - подчеркивания очень полезен для написания длинных имен и считается буквой. С него также может начинаться имя какой-либо переменной. Примеры идентификаторов (имен): sum, SUM, _sum, SuM, Vector_sum, Addition_vector. Программисты условились использовать в идентификаторах по возможности только строчные буквы, за исключением случаев, когда идентификатор именует константу. Формальное определение идентификатора (имени) имеет вид:
<идентификатор>::=[<буква>|<подчеркивание>]{<буква>|<подчеркивание>|<цифра>}.

*Ключевые слова*. Выше уже упоминалось о *ключевых (служебных)* словах. К ним относятся такие слова, за которыми в языке программирования закреплен определенный смысл или они имеют определенное назначение.

Их нельзя употреблять в других случаях кроме тех, для которых они предназначены. Количество ключевых слов не велико, их необходимо запомнить, а конкретное назначение каждого будет указывать по мере изложения основ программирования на СИ.

Таблица 2

*Список ключевых слов*

| | | | |
|---|---|---|---|
| *int* | *external* | *else* | *char* |
| *register* | *for* | *float* | *typedef* |
| *do* | *double* | *static* | *while* |
| *struct* | *goto* | *switch* | *union* |
| *return* | *case* | *long* | *sizeof* |
| *default* | *short* | *break* | *entry* |
| *unsigned* | *continue* | *uuto* | *if* |

Обратим внимание еще раз на то, что *ключевые слова* не могут быть использованы в качестве *имен* переменных (*идентификаторов*) так как они играют особую роль в языках программирования высокого уровня, включая и СИ.

***Комментарий*** **– пояснительный текст, который служит для документирования программы. Многострочный комментарий представляет собой набор символов, заключенный между /* и */. Однострочный комментарий – это набор символов, следующий после //.**

***Разделители*****. В СИ имеется некоторый набор символов, выполняющие как бы роль разделителей (ограничителей), за которыми закреплены определенные функции. Приведем их список:**

**( ) – вызов функции, порядок выполнения выражения;**

**[ ] – задает элемент массива;**

**{ } – задает составной оператор;**

**: – разделяет метку от оператора;**

**, – разделяет имена переменных в списке переменных;**

**; – разделяет операторы друг от друга.**

***Управляющие коды*** **– это последовательность символов управляющая выводом информации на печатающем устройстве (см. таблицу 3).**

Таблица 3

*Управляющие коды*

| Код | Назначение |
|---|---|
| \n | Новая строка |
| \t | Горизонтальная табуляция |
| \v | Вертикальная табуляция |
| \b | Возврат на символ (забой) |
| \r | Возврат в начало строки |
| \f | Прогон бумаги до конца страницы |
| \\ | Обратный слэш |
| \’ | Одинарная кавычка |
| \” | Двойная кавычка |
| \a | Звуковой сигнал |
| \? | Знак вопроса |
| \ddd | Код символа ASCII – от одной до трех восьмеричных цифр |
| \xhhh | Код символа ASCII – от одной до трех шестнадцатеричных цифр |

***Константы.*** **В отличие от переменных константы – объекты, которые в процессе выполнения программы не меняют своего значения. Их имена принято записывать заглавными буквами.**

**По типу принимаемых значений в СИ имеется четыре вида констант: целые, вещественные, символьные и строковые.**

***Целые константы*****. Данный вид констант может задаваться в десятичной, восьмеричной, шестнадцатеричной и двоичной системах счисления. Общая формула представления целого числа *N* в системе счисления, основанием которой является число *A*, имеет вид (1). Рассмотрим для примера константу, равную 125. Фактически ее мы записали в десятичной системе счисления, которая применяется для представления целых чисел в математике. Данная константа содержит три значащих разряда: единиц, десяток и сотен. Представим ее в двоичном коде. Мы должны определить значения коэффициентов $\alpha_i$ в выражении (1), которые могут принимать значения 0 или 1, таким образом, чтобы оно равнялось заданному числу. Выражение (1) в данном конкретном случае примет вид**

$$125 = \alpha_0 \cdot 2^0 + \alpha_1 \cdot 2^1 + \alpha_2 \cdot 2^2 + \alpha_3 \cdot 2^3 + \alpha_4 \cdot 2^4 + \alpha_5 \cdot 2^5 + \alpha_6 \cdot 2^6 \quad \textbf{(12)}$$

**Не имеет смысла учитывать следующие слагаемые, поскольку в них коэффициенты $\alpha_i$ будут равны 0, начиная с $\alpha_7$ . Подберем так значения коэффициентов $\alpha_i$, чтобы сумма (12) равнялась 125. В результате получаем**

$$125 = 1\cdot 1 + 0\cdot 2 + 1\cdot 4 + 1\cdot 8 + 1\cdot 16 + 1\cdot 32 + 1\cdot 64. \quad (13)$$

**Выписывая значения коэффициентов $\alpha_i$ в обратном порядке их следования в выражении (13), получаем значение константы 125 в двоичном коде**

$$1111101, \quad (14)$$

**которое содержит семь значащих разрядов.**

**Далее рассмотрим представление данной константы в восьмеричной и шестнадцатеричной системе счисления. В соответствии с формулой (1) для ее представления в восьмеричной системе счисления потребуется три разряда**

$$125 = \sum_{i=0}^{2} \alpha_i \cdot 8^i = \alpha_0 \cdot 8^0 + \alpha_1 \cdot 8^1 + \alpha_2 \cdot 8^2 = 5\cdot 1 + 7\cdot 8 + 1\cdot 64. \quad (15)$$

**В результате получаем o175.**

**Аналогичным образом представим эту же константу в шестнадцатеричной системе счисления. Применяя выражение (1), имеем**

$$125 = \sum_{i=0}^{1} \alpha_i \cdot 16^i = \alpha_0 \cdot 16^0 + \alpha_1 \cdot 16^1 = 1\cdot 1 + 8\cdot 16. \quad (16)$$

**Выполняя правила записи шестнадцатеричных чисел, для заданной константы получаем ox81.**

Таблица 4

*Формы представления константы 125*

| Система счисления | Двоичная | Восьмеричная | Шестнадцатеричная |
|---|---|---|---|
| Значение | 1111111 | o175 | ox81 |

Таким образом, константа 125, приведенная в десятичной системе счисления, в языке СИ может быть также представлена еще в трех видах таблица 4.

***Вещественные константы***. Константа с плавающей запятой (вещественная) – это действительное десятичное положительное число. Оно включает целую часть, дробную часть и экспоненту. Константы с плавающей запятой представ-ляются в следующей форме

[<цифры>][.<цифры>][<э>[-]<цифры>] . (17)

Здесь <цифры> – одна или более десятичных цифр от 0 до 9, <э> – признак экспо-нентности, задаваемая символами "E" или "e". Квадратные скобки обозначают, что фрагмент, заключенный в них, может, как присутствовать, так и отсутствовать в полном представлении вещественной константы. Следует учитывать такие прави-ла: целая или дробная часть могут быть опущены, но не обе сразу; точка с дробной частью или экспонента могут быть опущены, но не обе одновременно.

*Примеры* вещественных констант: 285.103, 2.85109E2, 285103e-3, 83., .28, .0028e2.

Константы с плавающей запятой всегда задаются положительными значениями. Если требуется ввести отрицательные значения, то они формируются с помощью константного выражения, состоящего из знака минуса и следующей за ним константы. Знак минус при этом рассматривается как арифметическая операция (унарный минус). *Например*: -.0025, -2.5e-3, -.125, -1.85E-8.

*Символьные константы*. Под символьной константой в языке СИ понимается буква, цифра, знак пунктуации или специальный символ, заключенный в апострофы. Значение такой константы равно коду, представляемого ею символа. Символьная константа представляется в следующей форме

'<символ>' ,

где <символ> – любой символ кроме ' (апострофа) или \ (обратного слеша). Они представляются следующим образом: '\'' – апостроф, '\\' – обратный слеш.

*Примеры* символьных констант: 'a', '!', '2', '\b'.

*Строковые константы*. Символьная строка – это последовательность символов, заключенная в двойные кавычки. *Символьная строка – массив символов* и она имеет следующую форму представления

"<символы>" ,

в котором <символы> – это произвольное количество символов, за исключением " " " и " \ ". Чтобы их использовать в строке, необходимо перед ними поставить дополнительный знак " \ ". *Примеры* строковых констант: "This is symbol of string\n", "Один \\ Два", "\"Хорошо\" – сказал он.", "" (пустая строка).

Для формирования символьных строк, которые занимают несколько строк в программе, используется комбинация символов обратный слеш и новая строка, которая представит в памяти ЭВМ строки как одну. Например, "Длинная строка является объединением нескольких строк.". В СИ строки неразделенные ничем представляются как одна строка (см. программу 1).

*Программа 1:*

```
void main (void)
 {
  char *s;
  s="Данный текст иллюстрирует"
    "возможность \nобъединения группы"
    "строк в одну длинную строку.\n"
    "Это необходимо для "
    "наглядности написания программ.\n";
  printf ("%s", s);
 }
```

Программа напечатает фрагмент текста в виде:

*Данный текст иллюстрирует возможность*
*объединения группы строк в одну длинную строку.*
*Это необходимо для наглядности написания программ.*

## *Операции*

Ранее отмечалось, что *переменные* это такие объекты, которые могут неоднократно изменять свои значения в ходе выполнения программы. Именно с помощью них можно получать и хранить в оперативной памяти ЭВМ значения интересующих программиста величин. Для того чтобы получать новые значения переменных, должны быть определены операции, проводимые над ними. В зависимости от типа принимаемых значений переменными над ними определяется тот или иной набор операций.

Операции определенные в СИ в порядке убывания приоритета (старшинства) представлены в следующей таблице

Таблица 5

*Назначение операций языка СИ*

| Операция | Назначение |
|---|---|
| [ ] | Задание элемента массива |
| ( ) | Вызов функции |
| . | Выбор поля структуры |
| -> | Выбор поля структуры с помощью указателя |
| ++, -- | Постфиксное/префиксное увеличение/уменьшение на 1. Если и то и другое встречается вместе в одном выражении, то постфиксное имеет более высокий приоритет |
| sizeof | Определение размера переменной в байтах |
| (тип) | Приведение к типу |
| ~ | Побитовое отрицание |
| ! | Логическое НЕ |
| - | Унарный минус |

| | |
|---|---|
| & | Определение адреса |
| * | Обращение по адресу |
| *, /, % | Умножение, деление и остаток от деления (приоритет одинаковый) |
| +, - | Сложение, вычитание (приоритет одинаковый) |
| <<, >> | Сдвиг влево, сдвиг вправо (одинаковый приоритет) |
| <, >, <=, >= | Сравнение (приоритет равный) |
| ==, != | Равно, не равно (приоритет равный) |
| & | Побитовое И |
| ^ | Побитовое исключающее ИЛИ |
| \| | Побитовое ИЛИ |
| && | Логическое И |
| \|\| | Логическое ИЛИ |
| ? : | Условный оператор |
| =, +=, -=, …, &= | Присваивания и замещения (равный приоритет) |
| , | Предписывает последовательность выполнения выражений |

Более подробно с каждой из операций познакомимся при изучении соответствующих типов данных.

***Скалярные типы данных***. По типу принимаемых значений переменные могут отличаться друг от друга, как и константы, занимаемой памятью и тем набором операций, которые можно применять к ним. Таким образом, необходимо по данным признакам отличать переменные и константы. В СИ это делается с помощью объявления типов.

По-существу определение типа переменной задается набором допустимых значений и набором действий, которые можно совершать над каждой переменной рассматриваемого типа.

В языке программирования СИ существует целый набор типов переменных, который можно разделить на два класса: первый – *базовые* (*скалярные*) типы (*int*, *float*, *char*, *double*, и т. д.), второй – *производные* (*сложные*) типы данных (*enum*, [ ], *struct*, *union* и другие). В СИ существует правило, что все переменные, используемые программой, должны быть описаны (т. е. для них должен быть определен однозначно тип принимаемых значений). Сейчас рассмотрим правила описания переменных базового типа.

***Переменные целого типа***. Переменные целого типа объявляются с помощью оператора описания имеющего вид

тип <идентификатор>[, < идентификатор >, . . .]; , (17)

где тип – одно из ключевых слов указывающих на тип принимаемых значений, <идентификатор> – имя переменной, которая принимает данный тип значений. Выражение, содержащееся в квадратных скобках, может, как присутствовать, так и отсутствовать. Оператор описания, как и любой другой оператор языка СИ, должен заканчиваться символом ";". В соответствии с этим правилом запишем вид оператора описания (объявления) переменных, которым даны имена *in*, *rama*, *cub*. Предполагается, что они принимают целочисленные значения. Тогда с учетом выражения (17) оператор объявления этих переменных имеет вид

int *in, rama, cub*;

Заметим, что между ключевым словом и первым именем переменной, по крайней мере, должно быть не менее одного пробела.

Типов переменных, принимающих целочисленные значения, в СИ не одно int. Это определяется оперативной памятью, отводимой для представления чисел. Объем памяти фактически определяет и диапазон значений, из которого переменная данного типа может принимать одно из значений. Типы целочисленных значений и их характеристики приведены в следующей таблице

*Таблица 1.2.5*

Характеристики целочисленных типов данных языка СИ

Таблица 6

*Характеристики целочисленных типов данных языка СИ*

| Тип | Объем памяти, байт | Диапазон значений |
|---|---|---|
| *int* | 2 | -32768 … 32767 |
| *unsigned* | 2 | 0 … 65535 |
| *short* | *int* | *int* |
| *unsigned short* | *unsigned* | *unsigned* |
| *long* | 4 | -2147483648 … 2147483647 |
| *unsigned long* | 4 | 0 … 4294967295 |

*Примеры* операторов объявления переменных, принимающие целочисленные значения различных типов:

*1)int* x, a, f;
*2)unsigned* p1, pi, I;
*3)short* k, m, n;
*4)long* p, t, r;
*5)unsigned long* q, qq, s2;

Над переменными, принимающими целочисленные значения, в СИ определены три типа операций: *арифметические, логические* и *битовые*. *Арифметические* операции в порядке убывания приоритета содержатся в таблице 7.

Таблица 7

*Арифметические операции*

| Операция | Назначение |
|---|---|
| + | Сложение, унарный плюс |
| - | Вычитание, унарный минус |
| * | умножение |
| / | Деление |
| % | Остаток от деления |
| x= | Изменить и заменить, где вместо символа x может быть одна из операций +, -, *, / или % |
| ++ | Инкремент (увеличить на 1) |
| -- | Декремент (уменьшить на 1) |

Обратим внимание читателя на то, что операция деления целого числа на целое представляется двумя операциями: деления на цело и вычисления остатка от деления. Рассмотрим конкретные примеры: 5 / 2 = 2 и 5 % 2 = 1.

В различных реализациях языка СИ при условии того, что хотя бы один из операндов является отрицательным числом, в результате выполнения операции % могут быть получены различные значения. Поэтому для получения мобильного (переносимого) кода рекомендуется применять эту операцию только для переменных, принимающих беззнаковые целочисленные значения, а именно для типов данных: *unsigned*, *unsigned short*, *unsigned long*. Для других же типов операнды должны принимать только положительные значения. Операция % неприменима к типам, принимающим значения с плавающей запятой. Рассмотрим примеры программ, выполняющие арифметические операции над переменными, которые принимают целочисленные типы данных.

*Программа 2:*

```
#include <stdio.h>
void main (void)
 {
  int x, r;
  unsigned y, z;
  x = -125;
  y = 29;
  r = -5;
  z = 3;
```

```
  x += r;  /* Эквивалентно оператору x = x + r */
  printf ("\n x = %d\n", x);
  x -= r;  /* Эквивалентно оператору x = x - r */
  printf ("x = %d\n", x);
  x = r;  / Эквивалентно оператору x = x * r */
  printf ("x = %d\n", x);
  x /= r;  /* Эквивалентно оператору x = x / r */
  printf ("x = %d\n", x);
  y %= z;  /* Эквивалентно оператору y = y % z */
  printf ("y = %d\n", y);
  y = ++z; /* Эквивалентно оператору y = z + 1 */
  printf ("y = %d   z = %d\n", y, z);
  y = z++; /* Эквивалентно оператору y = z */
  printf ("y = %d   z = %d\n", y, z);
 }
```

**В результате выполнения программы будут получены следующие данные:**

**x = -130**

**x = -125**

**x = 625**

**x = -125**

y = 2

y = 3  z = 3

y = 3  z = 4

Над переменными, принимающими целочисленные значения определены *логические* операции, представленные в следующей таблице

Таблица 8

*Логические операции*

| Операция | Назначение |
|---|---|
| && | И |
| \|\| | ИЛИ |
| ! | НЕ |
| == | Сравнение на равенство |
| != | Сравнение на неравенство |
| > | Сравнение на больше |
| >= | Сравнение на больше или равно |
| < | Сравнение на меньше |
| <= | Сравнение на меньше или равно |

**В СИ нет типа переменных, принимающих значения логического типа. Например, как в Паскале тип значений *boolean* или *logical* в Фортране. Однако определены логические и операции отношения для формирования и записи выражения, являющихся по смыслу принимаемых значений *логическими*. Они принимают значения либо 1 (*истина*) либо 0 (ложно). Данный тип принимаемых значений относится к типу *int*. Этот набор операций приведен в таблице 8. Предположим, что имеется две переменные X и Y, принимающие значение 0 или 1, а результаты действия этих операций содержатся в следующей таблице**

**Таблица 9**

***Результаты действия логических операций***

| X | Y | ! X | X && Y | X \|\| Y |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |

**Как видно из таблицы 9 результатом действия логических операций является целое число, принимающее значение 0 или 1.**

Другие операции, кроме трех рассмотренных только что, относятся к операциям *сравнения* выражений и (или) констант, принимающих целочисленные значения. По своему смысловому содержанию данные операции используются для записи неравенств, а в качестве ответа выступают либо 0 или 1 в зависимости от того верно или не верно неравенство.

*Примеры* выражений, содержащих операции сравнения:

1)5 <= 3;
2)(2*b + c) == 4;
3)da / fk + 2 != (c * (d * k – 3) + 3 * fk) – 4;
4)((d – 5 / fk) > (da / c + k) == (fk + d * c)) – 1.

Для целочисленных переменных также определены *битовые* операции (см. следующую таблицу).

Таблица 10

*Битовые операции*

| Операция | Назначение |
|---|---|
| & | И |
| \| | ИЛИ |
| ^ | Исключающее ИЛИ |
| ~ | Отрицание (НЕ) |

Таблица 10 (продолжение)

| Операция | Назначение |
|---|---|
| << | Сдвиг влево |
| >> | Сдвиг вправо |

Битовые операции обычно используются в приложениях тогда, когда возникает потребность для доступа к аппаратуре, манипулируя с битами на нижнем уровне. *Например*, такой способ часто применяется при управлении станками ЧПУ и другим оборудованием.

Чтобы понять, каким образом работают битовые операции, приведем таблицу, содержащую получаемые результаты. Для этого величины X и Y будут отражать значения в двух разрядах, над которыми проводятся битовые операции.

Таблица 11

*Результаты битовых операций*

| X | Y | ~ X | X ^ Y | X & Y | X \| Y |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |

Рассмотрим, каким образом работают битовые операции на конкретном примере, а именно для чисел 79 и 28. Для этого необходимо представить их в двоичном коде, в результате получим $79_2 = 1001111$ и $28_2 = 0011100$. Теперь над ними по разрядно проводим битовые операции:

1)79 & 28 = 1001111 & 0011100 = 0001100 = 12;
2)79 | 28 = 1001111 | 0011100 = 1011111 = 95;
3)79 ^ 28 = 1001111 ^ 0011100 = 1010011 = 83;
4)~ 79 = ~ 1001111 = 0110000 = 48;
5)~ 28 = ~ 0011100 = 1100011 = 99.

Отдельно рассмотрим операции сдвига по разрядной сетке влево (<<) и вправо (>>). Предположим, что необходимо осуществить сдвиг на 2 разряда. Естественно данные операции проводятся над числами в двоичной системе счисления. Получим следующие результаты:

1)79 >> 2 = 1001111 >> 2 = 0010011 = 19 (эквивалентно $79 / 2^2$);
2)28 >> 2 = 0011100 >> 2 = 0000111 = 7 (эквивалентно $28 / 2^2$);
3)79 << 2 = 1001111 << 2 = 100111100 = 316 (эквивалентно $79 * 2^2$);
4)28 << 2 = 0011100 << 2 = 001110000 = 112 (эквивалентно $28 * 2^2$).

При проведении операции сдвига вправо нижние разряды теряются, а в верхних разрядах автоматически выставляется значение 0. Когда же проводится операция сдвига влево, то в добавляемых нижних разрядах автоматически выставляется значение 0.

***Переменные вещественного типа*. В большинстве прикладных задач, которые решаются с помощью ЭВМ, часто приходится иметь дело с величинами, предс-тавляющиеся *дробными* (вещественными) числами. Для таких объектов (пере-менных и констант) в СИ предусмотрен вещественный тип значений. Перемен-ные вещественного типа, как и любого другого типа должны быть объявлены и для этого используются ключевые слова *float*, *double* и *long double*. Это указы-вает на то, что в СИ имеется три типа вещественных данных, которые отличают-ся друг от друга объемом памяти, отводимой на хранение соответствующего значения, и, следовательно, диапазоном принимаемых значений. Оператор объявления переменных вещественного типа согласно выражению (17) будет иметь одну из форм**

*float* <идентификатор>[, < идентификатор >, . . .];
*double* <идентификатор>[, < идентификатор >, . . .];
*long double* <идентификатор>[, < идентификатор >, . . .]; .

**Основные характеристики объектов вещественного типа содержатся в таблице**

Таблица 12

*Характеристики вещественных типов данных языка СИ*

| Тип | Объем памяти, байт | Диапазон значений |
|---|---|---|
| *float* | 4 | 3.4E-38 … 3.4E38 |
| *double* | 8 | 1.7E-308 … 1.7E308 |
| *long double* | 10 | - |

*Примеры* операторов объявления переменных, принимающих вещественные значения. Так, например, для хранения значения переменных с именами f1, f2, ff должно быть отведено по 4 байта, а для переменных tt, tr, ts, te – 8 байтов и для объектов mmq, mwq по 10 байтов. Соответствующие операторы объявления имеют вид:

*float* f1, f2, ff;

*double* tt, tr, ts, te;

*long double* mmq, mwq; .

Для значений вещественного типа определены две группы операций: *арифметические* и *логические*. Для них в отличие от целочисленных типов данных определена одна операция деления (она обозначается символом " / "), результатом которой является также значение вещественного типа.

В качестве *примера* приведем листинг программы для решения следующей задачи: требуется вычислить площадь кольца, у которого радиусы внешней и внутренней окружности равны r1 и r2 соответственно.

*Программа 4:*

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define PI 3.1415926535
void main (void)
 {
  float r1, r2, s;
  clrsrc ( );
  printf ("Введите значения r1, r2: ");
  scanf ("%f %f",&r1,&r2);
  s = PI * (pow(r1,2) - pow(r2,2));
  printf ("\nПлощадь кольца s = %f ", s);
 }
```

Битовые операции для вещественных переменных не имеют смысла, так как их значения не возможно точно представить в двоичной системе счисления. Они представляются приближенно, используя конечное число разрядов.

***Переменные символьного типа***. Как правило, большинство программ, написанных на любом языке программирования (не является исключением и язык СИ), имеет дело с символами, которые применяются для представления информации в виде текста. По этой причине в СИ предусмотрен символьный тип значений, используемый для объявления переменных символьного типа. ***Символьные*** типы значений в СИ бывают двух видов и обозначаются ключевыми словами *char* и *unsigned char*. Для хранения их значения компилятором отводится 1 байт памяти, что и определяет диапазон принимаемых значений таблица 13

Таблица 13

*Характеристики символьных типов данных*

| Тип | Объем памяти, байт | Диапазон значений |
|---|---|---|
| *char* | 1 | -128 … 127 |
| *unsigned char* | 1 | 0 … 255 |

В программах они объявляются с помощью операторов вида:

*char* <идентификатор>[, < идентификатор >, . . .];

*unsigned char* <идентификатор>[, < идентификатор >, . . .]; .

По существу они являются подмножеством целых и поэтому их можно свободно смешивать в выражениях с переменными типа *int*.

Для примера приведем программу, которая оперирует одновременно с переменными символьного и целого типов.

*Программа 6:*

```
#include <stdio.h>
#include <conio.h>
void main (void)
 {
  int j, k[];
  char b[];
  clrsrc ( );
  j = 0;
  while (b[j] != '\0')
   {
    k[j] = b[j];
    ++j;
   };
  j = 0;
  while (k[j] != '\0')
   printf ("%d\n", k[j]);
 }
```

***Перечислимый тип данных***. Не редки случаи, когда приходится иметь дело с объектами (переменными), которые могут принимать перечислимое множество значений, например: дни недели (понедельник, вторник, … , воскресенье), месяцы года (январь, февраль, … , декабрь), номенклатура изделий завода или фирмы. При этом хотелось бы, чтобы программа имела дело с понятными значениями, а именно с принятыми названиями. Так, например дни недели можно пронумеровать и программа будет оперировать с целыми числами. Способы нумерации у каждого программиста могут быть свои, а для удобства восприятия программы хотелось бы использовать общепринятые обозначения. Для таких целей в СИ предусмотрен перечислимый тип *enum*, который необходимо вводить (объявлять) с помощью оператора, имеющего одну из форм записи

*enum* [<тег>]{<список перечисления>} < описатель >[, < описатель >. . .];
*enum* <тег> <идентификатор>[, < идентификатор >, . . .]; .

Здесь <тег> – идентификатор, именующий перечислимый тип, который специфицируется данным списком; <список перечисления> представляет собой одну или более конструкций вида <идентификатор>[= < константное выражение>], разделяемые друг от друга запятой; <идентификатор> – имя переменной перечислимого типа.

***Примеры*** **операторов объявления переменных перечислимого типа:**

***enum*** **{пон, вт, ср, четв, пят, суб, воскр} day_week;**

***enum*** **week_type {пон, вт, ср, четв, пят, суб, воскр};**

***enum*** **week_ type day_ week; .**

**Пример программы, которая выводит на экран монитора название дней недели, используя переменные перечислимого типа.**

***Программа 7:***

```
#include <stdio.h>
#include <conio.h>
void main (void)
 {
  enum {пон, вт, ср, четв, пят, суб, воскр} day;
  clrsrc ();
  printf ("Введите значение day (0<=day<=6):");
  scanf ("%d", &day);
  switch (day)
   {
    case пон:
    printf ("Понедельник");
    break;
```

```
case вт:
printf ("Вторник");
break;
case ср:
printf ("Среда");
break;
case четв:
printf ("Четверг");
break;
case пят:
printf ("Пятница");
break;
case суб:
printf ("Суббота");
break;
case воскр:
printf ("Воскресенье");
break;
```

```
    default:
    printf ("Переменная day не может принимать данного
значения");
   }
 }
```

***К О Н Е Ц***