

**C/C++**

Язык **C** был разработан Деннисом М. Ритчи в 1972 году. Реализация **C**, в соответствии с изложенными правилами, рассматривается как K&R стандарт (Брайн В. Керниган и Ритчи). K&R – минимальная стандартная реализация. В 1983 году Американский институт национальных стандартов (ANSI) разработал новый стандарт, названный стандартом ANSI языка **C**.

**C++** - это надмножество языка **C**. Реально он включает все операторы и средства языка **C**, добавив только некоторые новые.

Преимущество **C++** в том, что он позволяет с большой легкостью разрабатывать большие сложные программы за счет модульного подхода и некоторых других усовершенствований. Кроме того, **C++** является языком ООП.

# Логические структуры языка.

Программа состоит из лексических элементов.

## 1. Элементы

Программа на С представляет собой строки, состоящие из лексических элементов пяти типов: ключевые слова, константы, операции, ограничители, идентификаторы.

Смежные элементы отделяются друг от друга разделителями или комментариями.

Разделители: пробелы, символы табуляции, возврата каретки, перевода строки.

## 2. Комментарии

Они служат для документирования программы

Формат **/\* текст комментария \*/**

### Пример

**/\* Программа выводит сообщение на**

### 3. Ограничители

Символы – ограничители: ( ), [ ], { }, : , , , ;

### 4. Операции

Перечень операций приведен ниже:

Одно символьные операции: = ! ^ & \* - : . < > /  
? + %

Двух символьные операции: == != && :: -> ++ --  
<< >> <= >= += -= \*= /= %= ^= := &=

Трехсимвольные операции: <<= >>=

### 5. Идентификаторы

Они служат для именованя типов, переменных, констант и функций.

Идентификатор состоит из букв и цифр и может содержать символы подчеркивания.

Значащие – первые 32 символа. Начинается идентификатор с буквы. В идентификаторах прописные и заглавные буквы различаются.

## 6. Зарезервированные слова

В С используются зарезервированные слова, которые нельзя использовать в качестве идентификаторов. Они задаются прописными буквами. Фрагмент таблицы зарезервированных слов.

auto	double	goto	main	switch	while
break	else	if	pointer	typedef	
case	float	long	return	union	

# Константы

В С имеется четыре типа констант: целые, вещественные, символьные и строковые.

## Константы целого типа

Константы целого типа могут задаваться в десятичной, 8-ой или 16-ой системах счисления.

Десятичные целые константы образуются из цифр. Первой цифрой не должен быть 0.

Восьмеричные константы начинаются с цифры 0, за которой следуют цифры 0-7.

Шестнадцатеричные константы начинаются с цифры 0 и символа  $\left\{ \begin{matrix} x \\ X \end{matrix} \right\}$ , за которыми может стоять одна или более 16-ых цифр 0-9, A-F.

Пример

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    int a=3478,
```

```
    b=06626,
```

```
    c=0xD96;
```

```
    printf("a=%d b=%d c=%d\n",a,b,c);
```

```
}
```

На экране

```
a=3478 b=3478 c=3478
```

## Константы вещественного типа

Константы этого типа состоят из цифр, десятичной точки и знака десятичного порядка

$\left. \begin{array}{l} e \\ E \end{array} \right\}$

Примеры

**1. 2e1 .1234 .1e3**

**.1 2E1 1.234 0.0035e-6**

**1.0 2e-1 2.1e-12 0.234**



## Символьные константы

Символьные константы заключаются в одиночные кавычки (апострофы)

Например

```
if (ch>='a'&&ch<='z')
```

Одни символьные константы соответствуют символам, которые можно вывести на экран, другие – управляющим символам, задаваемым с помощью **esc**-последовательности, третьи – форматизирующим символам, также задаваемым с помощью **esc** – последовательности.

Например

Символ “апостроф” → ‘\’;

Переход на новую строку → ‘\n’;

Обратный слэш → ‘\\’.

# Управляющие коды

В следующей таблице приведены управляющие коды. Каждая **esc** – последовательность должна быть заключена

esc-последовательность	Назначение
\n	Новая строка
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\b	Возврат на символ
\r	Возврат в начало строки
\f	Прогон бумаги до конца страницы
\\	Обратный слеш
\'	Одинарная кавычка
\"	Двойная кавычка
\a	Звуковой сигнал
\ddd	Код символа в ASCII-от 1 до 3-х восьмеричных цифр
\xhhh	Код символа в ASCII-от 1 до 3-х шестнадцатеричных цифр

# Строковые константы

Строковые константы состоят из нуля или более символов, заключенных в двойные кавычки. В строковых константах управляющие коды задаются с помощью **esc**-последовательности.

## Замечания к использованию констант

Для задания констант можно использовать их непосредственное написание (обозначение). Кроме того, существуют следующие способы задания константы:

а) Макроопределение

Формат

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

Например

```
#define PI 3.14
```

```
#define CHARACTER_B 'B'
```

```
#define version_oct 020 // для восьмеричного числа
```

```
#define version_dec 16 // для десятичного числа
```

```
#define version_hex 0x10 // для 16-го числа
```

```
#define NAME "ALEKS"
```

а) Макроопределение

Формат

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

Например

```
#define PI 3.14
```

```
#define CHARACTER_B 'B'
```

```
#define version_oct 020 // для  
    восьмеричного числа
```

```
#define version_dec 16 // для десятичного  
    числа
```

```
#define version_hex 0x10 // для 16-го числа
```

```
#define NAME "ALEKS"
```

## б) типизированные константы

Можно определить константу, описать тип данных и присвоить значение, используя ключевое слово **const**

Например

```
main()
```

```
{
```

```
    const int CHILDREN=8;
```

```
    const char INIT='C';
```

```
    const float NUMBER=1.65;
```

```
}
```

# Скалярные типы данных, операции, преобразование типов.

Используются различные типы данных для представления хранимой и обрабатываемой информации.

## Типы данных и элементы памяти.

Тип задается набором допустимых значений и набором действий, которые можно совершать над каждой переменной рассматриваемого типа. Переменные типизируются посредством их описаний. Выражения типизируются посредством содержащих в них операций.

Есть predetermined типы данных: целые, вещественные, указатели, переменные, массивы, функции, объединения, структуры и **void** (отсутствие типа).

**Скалярные типы:** указатель,

арифметический (целый, вещественный)

# Агрегатные типы: массив, структура, объединение.

В следующей таблице представлены типы C, их размеры и диапазоны

Тип	Размер в байтах	Диапазон значений
char, signed char	1	-128...127
unsigned char	1	0...255
int, signed int, short int, signed short int	2	-32768...32767
unsigned int, unsigned short int	2	0...65535
long int, signed long int	4	-2147483648...2147483647
unsigned long int	4	0...4294967295
float	4	3.4E-38...3.4E38
double	8	1.7E-308...1.7E308
long double	10	3.4E-4932...3.4E4932
pointer	2	
pointer	4	

## 2. Переменные целого типа

Переменная описывается с помощью спецификатора типа (см. таблицу) и при описании ей может быть присвоено начальное значение.

Пример

```
int age=20, height=170; // возраст, рост  
unsigned weight=height/2; // вес  
long index; // индекс
```

Замечание Если используются спецификаторы **unsigned**, **short**, **long**, то **int** можно опускать.

Допустимые операции над целочисленными операндами указаны в таблице



## Арифметические операции

Обозначение	Операция
+	Унарный плюс, сложение
-	Унарный минус, вычитание
*	Умножение
/	Деление
%	Остаток от деления
x=	Изменить и заменить, где $x = \{+, -, *, /, \%\}$
++	Инкремент (увеличение на 1)
--	Декремент (уменьшение на 1)

## Логические операции

Обозначение	Операция
&&	И (логическое умножение)
	ИЛИ (логическое сложение)
!	НЕ (отрицание)
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

## Битовые операции

Обозначение	Операция
&	И (and)
	ИЛИ (or)
^	ИСКЛЮЧАЮЩЕЕ ИЛИ
~	Отрицание
>>	Сдвиг вправо
<<	Сдвиг влево
x =	Изменить и заменить, где $x = \{ \&,  , ^, >>, << \}$

### 3. Переменные вещественного типа

Для описания таких переменных используются спецификаторы **float**, **double**, **long double**.

Пример

**float force=12.78, /\* сила \*/**

**acceleration=1.234; /\* ускорение \*/**

**double height; /\* высота \*/**

Операции над вещественными операндами, аналогичны арифметическим и логическим операциям над целочисленными операндами (см. предыдущую таблицу). Исключение – операция % (остаток от деления).

## 4. Символьные переменные

Для описания символьных переменных используются спецификаторы **char**, **signed char**, **unsigned char**.

Можно задавать начальные значения.

Пример

```
char ch='$', ans='n', ascii_value=65;
```

### Замечание

В выражениях переменные типа **char** могут смешиваться с переменными типа **int**, поскольку те и другие принадлежат к целому типу.

## Пример

```
#include <stdio.h>  
main()  
{  
char ch='a', ans;  
printf("значение ch+3=%c",ch+3);  
ans=ch%3;  
printf(("\\n\\n значение ans=%d\\n",ans);  
}
```

Программа выводит на экран следующие строки:

**Значение ch+3=d**

**Значение ans=1**

## 5. Строковые переменные

Для определения строковой переменной необходимо использовать тип **char** и указать максимальное число символов, которое может содержать строка. Строка объявляется, как массив символов, но для работы с массивом символов, как со строкой имеется набор библиотечных функций.

Описание в общем случае:

**char name[n];**

**name** – имя массива;

**n** – размер массива.

Пример **char str[80];**

Работу со строковыми данными рассмотрим подробнее при изучении массивов и функций работы со строками.

## **6. Приоритет и порядок выполнения операций.**

Если в выражении не используются круглые скобки, задающие порядок выполнения операций, то группировка операндов для операций производится с учетом приоритета операций. В следующей таблице приведены операции в порядке убывания приоритета.

Операция	Назначение
[]	Задание элемента массива
()	Вызов функции
.	Выбор поля структуры
->	Выделение поля структуры с помощью указателя
++, --	Постфиксное/префиксное увеличение на 1 (постфиксное более приор.)
sizeof	Определение размера переменной в байтах
(тип)	Приведение к типу
~	Побитовое отрицание
!	Логическое НЕ
-	Унарный минус
&	Определение адреса
*	Обращение по адресу
*, /, %	Умножение, деление, остаток
+, -	Сложение, вычитание
<<, >>	Сдвиг влево, сдвиг вправо
<, >, <=, >=	Сравнение
=, !=	Равенство, неравенство
&	Побитовое И
^	Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ
	Побитовое ИЛИ
&&	Логическое И
	Логическое ИЛИ
?:	Условный оператор
=, +=, -=, *=, /=, %=, &=, &=, ^=,  =, <<=, >>=	Присваивание и замещение
,	Операция запятая, которая предписывает последовательное вычисление выражений



## 7. Операции отношения

Логические выражения строятся из операций отношения и вырабатывают в качестве результата значение типа **int**. Если результат равен 0, то считается, что логическое выражение ложно, в противном случае – истинно. Если в логическом выражении не используются скобки, то оно вычисляется слева направо. Вычисление прекращается, как только результат становится определенным. Такой способ вычисления логических выражений называется усечением.

Например

**if (a>b || c>d || e>f)**

Вычисление выражения прерывается, как только выясняется, что, либо **a>b**, либо **c>d**, либо **e>f**.

Усечение с успехом может быть использовано для задания корректного порядка вычислений в логических выражениях.

Например, логическое выражение

**if (b!=0.0&&a/b>12.4)**

Имеет больший смысл, чем логическое выражение

**if (a/b>12.4&&b!=0.0)**

## 8. Побитовые операции

Операнд или операнды побитовых операций должны быть целого типа. Побитовые операции используются для манипуляция с битами на нижнем уровне. Операции  $\&$ ,  $|$ ,  $\wedge$  определяются следу

<b>И</b>	<b>0</b>	<b>1</b>
<b>0</b>	<b>0</b>	<b>0</b>
<b>1</b>	<b>0</b>	<b>1</b>

<b>ИЛИ</b>	<b>0</b>	<b>1</b>
<b>0</b>	<b>0</b>	<b>1</b>
<b>1</b>	<b>1</b>	<b>1</b>

<b>ИСК. ИЛИ</b>	<b>0</b>	<b>1</b>
<b>0</b>	<b>0</b>	<b>1</b>
<b>1</b>	<b>1</b>	<b>0</b>

Операции  $\ll$  и  $\gg$  служат для сдвига последовательности битов, соответственно, влево и вправо.

Эти операции можно применять для деления или умножения на число, равное степени 2, в соответствии со следующими

правилами:

$x \gg n$  – эквивалентно делению  $x$  на  $2^n$

$x \ll n$  – эквивалентно умножению  $x$  на  $2^n$

**Пример**

$123 = 0000000001111011$

$123 \ll 5 \rightarrow 0000111101100000 = 3936$

$123 \gg 1 \rightarrow 00000000000111101 = 61$

## 9. Операции присваивания

К ним относятся:  $=$  ,  $+=$  ,  $-=$  ,  $*=$  ,  $/=$  ,  $\%=$  , и префиксные и постфиксные операции  $++$  и  $--$ .

Все операции присваивают переменной результат вычисления выражения. Если тип левой части отличается от типа правой части, то тип правой части приводится к типу левой. В одном операторе операция присваивания может встречаться несколько раз. Вычисление производится справа налево. Например  $a=(b=c)*d$ ; Значению  $b$  присваивается значение  $c$ , затем выполняется операция умножения на  $d$  и результат присваивается  $a$ .

Типичный пример использования многократного присваивания  $a=b=c=d=e=f=0$ ;

Операции  $+=$  ,  $-=$  ,  $*=$  ,  $/=$  укороченная форма, т.е.

$a+=b$ ;  $\rightarrow a=a+b$ ;  $a-=b$ ;  $\rightarrow a=a-b$ ;  $a*=b$ ;  $\rightarrow a=a*b$ ;  
 $a/=b$ ;  $\rightarrow a=a/b$ ;

Постфиксные и префиксные операции ++ и -- используются для увеличения (инкремент) и уменьшения (декремент) на 1 значения переменной. Семантика операций следующая:

**++a , a++** – увеличение значения переменной **a** на 1 до (после) ее использования в выражении.

**--a , a--** – уменьшение значения переменной **a** на 1 до (после) ее использования в выражении.

Пример

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    int a,b,c=3,d=4,f=3,g,h=5,z=6,i;
```

```
    a=z+(b=c*d*f+(g=h+(i=3)));
```

```
    printf(“%d\n”,a);
```

```
} // Программа выводит на экран значение – ?.
```

## 10. Операция `sizeof`

Эту операцию можно применять к константе, типу или переменной. В результате будет получено число байтов, занимаемых операндом. Если операндом является тип, то такой операнд следует заключать в круглые скобки. Если операнд переменная, то скобки можно опускать.

Пример

```
#include <stdio.h>
```

```
main() {
```

```
float a;
```

```
int b;
```

```
char c;
```

```
float d[500];
```

```
printf(“\n Размер памяти под целое %d”,  
    sizeof(int));  
printf(“\n Размер памяти под двойную  
    точность %d”, sizeof(double));  
printf(“\n Размер памяти под переменную  
    a %d”, sizeof a);  
printf(“\n Размер памяти под массив d  
    %d”, sizeof d); }
```

Результат работы программы

**Размер памяти под целое 2**

**Размер памяти под двойную точность 8**

**Размер памяти под переменную a 4**

**Размер памяти под массив d 2000**



## 11. Операция “запятая”

Операция “запятая” для связывания между собой выражений. Список выражений, разделенный запятыми, трактуется как единое выражение и вычисляется слева направо. Пример

**if (c=getchar(),c>'a')** Читается символ в переменную **c** и сравнивается с символом **'a'**, результат целое число.

## 12. Приведение и преобразование типов

Для выполнения однозначного преобразования (cast) объектов одного типа в другой в С имеется специальная конструкция вида:

**(имя типа) выражение:**

Преобразование одного типа в другой тип данных выполняется в соответствии со следующими условиями:

преобразование используется для однозначного перевода данного значения в другой тип;

операнд автоматически приводится к другому типу перед выполнением соответствующей арифметической или логической операции;

если операнд одного типа присваивается левому допустимому объекту другого типа, то приведение типов выполняется автоматически;

аргумент функции и результат может <sup>34</sup>автоматически приводиться к требуемому