Основы алгоритмизации и программирования

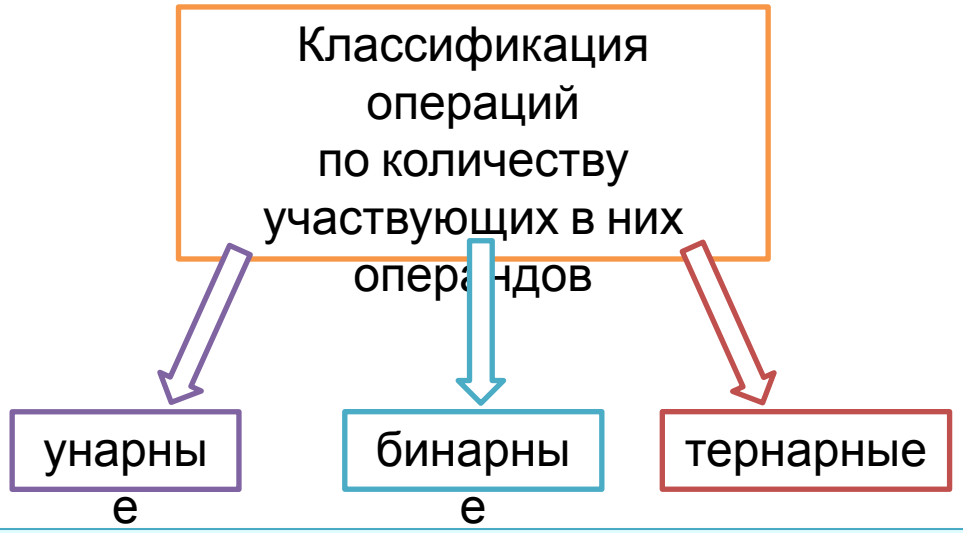
Лекция 4

Операции и выражения языка Си

Обзор операций и выражений

Выражения используются для вычисления значений (определенного типа) и состоят из **операндов**, **операций** и **скобок**. Каждый операнд может быть, в свою очередь, выражением или одним из его частных случаев – константой или переменной. Операнды задают данные для вычислений.

Знак операции – это один или более символов, определяющих действие над операндами, т.е. операции задают действия, которые необходимо выполнить. Внутри знака операции пробелы не допускаются.



Операции выполняются в соответствии с **приоритетами**. Для изменения порядка выполнения операций используются **круглые скобки.**

Большинство операций выполняются **слева направо**, например, ***a+b+c*** → ***(a+b)+c***.

Исключение составляют **унарные операции**, **операции присваивания** и **условная операция (?:)**, которые выполняются **справа налево**.

Обзор операций и выражений

В языке **Си** используются четыре унарные операции, имеющие **самый высокий приоритет**, их часто называют первичными.

Первичные операции

Операция доступа к полям структур и объединений при помощи идентификаторов **«.»** – **точка**

Операция доступа к полям структур и объединений при помощи указателей **«–>»** – **стрелка**

Операция **[ ]** индексации, используемая при декларации массива и обращении к его элементам

Операция **( )** обращения к функции

Арифметические операции

Обозначения арифметических операций

+	сложение	-	вычитание
/	деление ★	*	умножение
%	остаток от деления целочисленных операндов со знаком первого операнда – деление «по модулю»		

★ Деление, для **int** операндов происходит с отбрасыванием остатка

Обзор операций и выражений

Операндами традиционных арифметических операций **(+ – * /)** могут быть **константы**, **переменные**, **обращения** к возвращающим значения **функциям**, **элементы массивов**, любые **арифметические выражения**, **указатели** (с ограничениями)

Порядок выполнения действий в арифметических выражениях

Выражения в круглых скобках

операции * , / , %

операции + , –

Операции * , / , % обладают **высшим приоритетом** над операциями +, –, поэтому при записи сложных выражений нужно использовать общепринятые математические правила - использовать круглые скобки.

$$x + y \cdot z - \frac{a}{b+c} \leftrightarrow x+y*z-a/(b+c)$$

Обзор операций и выражений

Формат операции присваивания

Операнд_1 = Операнд_2 ;

Операндом_2 (**правый операнд**) могут быть: **константа**, **переменная** или **любое выражение**, составленное в соответствии с синтаксисом языка **Си**. Правый операнд операции присваивания назвали **R–значение**, (**R–value, Right–value**).

Операция присваивания

Операндом_1 (**левый операнд**) может быть только **переменная**.
Левый операнд операции присваивания получил название **L–значение**, (***L–value, Left–value***) – **адресное выражение**.
Так в **Си** называют любое выражение, адресующее некоторый участок оперативной памяти, в который можно записать некоторое значение.
Переменная – это частный случай адресного выражения.

Присваивание значения в языке **Си**, в отличие от традиционной интерпретации, рассматривается как выражение, имеющее значение левого операнда после присваивания. Таким образом, присваивание может включать несколько операций присваивания, изменяя значения нескольких операндов

```
int i, j, k;
float x, y, z;
  ...
i = j = k = 0;   ↔   k = 0, j = k, i = j;
```

Примеры недопустимых выражений:
– присваивание константе: **2 = *x* + *y*;**
– присваивание функции: ***getch*() = *i*;**
– присваивание результату операции: **(*i* + 1) =2 + *y*;**

Обзор операций и выражений

Сокращенная запись операции присваивания

В языке Си используются два вида сокращенной записи операции присваивания

Вместо записи: ***v = v # e;***
где **#** – любая арифметическая операция (операция над битовым представлением операндов), рекомендуется использовать запись

v #= e;

Например:

i = i + 5; ↔ ***i += 5;***

(***знаки операций – без пробелов***)

Вместо записи: ***x = x # 1;***
где **#** – символы, обозначающие операцию инкремента **(+1)**, либо декремента **(–1)**, ***x*** – целочисленная переменная (или переменная-указатель), рекомендуется использовать запись:

##x; – **префиксную**,

или

x##; – **постфиксную**

Если эти операции используются в чистом виде, то различий между постфиксной и префиксной формами нет. Если же они используются в выражении, то в префиксной форме (***##x***) сначала значение **x** изменится на **1**, а затем полученный результат будет использован в выражении; в постфиксной форме (***x##***) – сначала значение переменной **x** используется в выражении, а затем изменится на **1**

Обзор операций и выражений

Пример 1:		Пример 2:	
int i, j, k;	Смысл записи	int n, a, b, c, d;	Значения
float x, y;		n = 2; a = b = c = 0;	
...		a = ++n;	n=3, a=3
x *= y;	x = x*y;	a += 2;	a=5
i += 2;	i = i + 2;	b = n++;	b=3, n=4
x /= y+15;	x = x/(y + 15);	b –= 2;	b=1
--k;	k = k – 1;	c = --n;	n=3, c=3
k--;	k = k – 1;	c *= 2;	c=6
j = i++;	j = i; i = i + 1;	d = n--;	d=3, n=2
j = ++i;	i = i + 1; j = i;	d %= 2;	d=1

Преобразование типов операндов арифметических операций

В **Си** различают **явное** и **неявное** преобразование типов данных. **Неявное** преобразование типов данных выполняет компилятор, а явное преобразование данных выполняет сам программист. **Результат любого вычисления будет преобразовываться к наиболее точному типу данных, из тех типов данных, которые участвуют в вычислении**.

x	y	Результат деления	Пример
делимое	делитель	частное	x = 15 y = 2
int	int	int	15/2=7
int	float	float	15/2=7.5
float	int	float	15/2=7.5

Преобразование типов операндов арифметических операций

Если операнды арифметических операндов **имеют один тип**, то и результат операции будет иметь **такой же тип**

Стрелки отмечают преобразования даже однотипных операндов перед выполнением операции

Но, как правило, в операциях участвуют операнды различных типов. В этом случае они **преобразуются** к общему типу в порядке увеличения их **«размера памяти»**, т.е. объема памяти, необходимого для хранения их значений. Поэтому неявные преобразования всегда идут от «меньших» объектов к «большим».

short, char	→ int	→ unsigned	→ long	→ double
			float	→ double

Действуют следующие правила

Если один из операндов имеет тип **double**, то и другой преобразуется в **double**

Если один из операндов **long**, то другой преобразуется в **long**

Значения типов **char** и **short** всегда преобразуются в **int**

Явное преобразование типов

В любом выражении преобразование типов может быть осуществлено **явно**, для этого достаточно перед выражением поставить в круглых скобках атрибут соответствующего типа

(тип) выражение;

Операция приведения типа вынуждает компилятор выполнить указанное преобразование, но ответственность за последствия возлагается на программиста.

Использовать эту операцию рекомендуется везде, где это необходимо, например:

```
double x;
int n = 6, k = 4;
x = (n + k)/3;          //  x = 3, т.к. дробная часть будет отброшена;
x = (double)(n + k)/3;  //  x = 3.333333 – использование операции приведения типа позволило избежать округления результата деления целочисленных операндов.
```

Операции сравнения

Операции сравнения в языке Си

==	равно или эквивалентно	!=	не равно
<	меньше	<=	меньше либо равно
>	больше	>=	больше либо равно

Общий вид операций

Операнд_1 ***Знак операции*** ***Операнд_2***

В языке Си нет логического типа данных. Результат операции отношения имеет значение **1**, если отношение истинно (**true**), или **0** – в противном случае, т.е. – ложно *(***false***)*. Следовательно, операция отношения может использоваться в любых арифметических выражениях.

Указанные операции выполняют **сравнение значений** первого операнда со вторым. Операндами могут быть **любые арифметические выражения** и **указатели**. **Значения** арифметических выражений перед сравнением **вычисляются** и **преобразуются к одному типу**

Пример

y **> 0 ,** ***x*** **==** ***y*** **,** ***x*** **!= 2**

Логические операции

Логические операции в порядке убывания относительного приоритета

!	отрицание (логическое «НЕТ», “NOT”)
&&	конъюнкция (логическое «И», “AND”)
||	дизъюнкция (логическое «ИЛИ», “OR”)

Операндами (**выражениями**) логических операций могут быть **любые скалярные типы**. Ненулевое значение операнда трактуется как «**истина**», а нулевое – «**ложь**». Результатом логической операции, как и в случае операций отношения, может быть **1** или **0**.

Операция отрицания:

!0 → 1

!5 → 0

x = 10;

! (*x* > 0) → 0

Общий вид операции ***отрицания:***

! выражение

Логические операции

Общий вид операций ***конъюнкции*** и ***дизъюнкции***:

Выражение_1 знак операции Выражение_2

если **выражение_1** операции «**конъюнкция**» **ложно**, то результат операции – **ноль** и **выражение_2 не вычисляется**

если **выражение_1** операции «**дизъюнкция**» **истинно**, то результат операции – **единица** и **выражение_2 не вычисляется**

Пример

y **> 0 &&** ***x*** **= 7** → **истина, если оба выражения истинны;**
e **> 0 || *x* = 7** → **истина, если хотя бы одно выражение истинно**.

Старшинство операции «**И**» выше, чем «**ИЛИ**» и обе они младше операций отношения и равенства. **Относительный приоритет** логических операций позволяет пользоваться общепринятым математическим стилем записи сложных логических выражений, например:
0 < *x* < 100 ↔ 0 < *x* && *x* < 100 ;
x **> 0, *y* ≤ 1 ↔ *x* > 0 && *y* <=1 .**
Учет этих свойств очень существенен для написания правильно работающих программ.

Побитовые логические операции

В языке **Си** предусмотрен набор операций для работы с **отдельными битами**. Эти операции **нельзя применять к переменным вещественного типа**

Операции над битами

~	**дополнение (унарная операция); инвертирование (одноместная операция)**
&	**побитовое «И» – конъюнкция**
|	**побитовое «ИЛИ» – дизъюнкция**
^	**побитовое исключающее «ИЛИ» – сложение по модулю 2**
^	**побитовое исключающее «ИЛИ» – сложение по модулю 2**
>>	**сдвиг вправо**
<<	**сдвиг влево**

Общий вид операции инвертирования (поразрядное отрицание):

~ выражение

инвертирует каждый разряд в двоичном представлении своего операнда

Остальные операции над битами имеют вид:

Выр_1 ***знак операции*** ***Выр_2***

Побитовые логические операции

Операндами операций над битами могут быть только **выражения**, приводимые к **целому типу**. Операции (**~, &, |, ^**) выполняются **поразрядно** над всеми битами операндов (**знаковый разряд особо не выделяется**)

Пример

~0*x*F0	↔	*x*0F
0*x*FF & 0*x*0F	↔	*x*0F
0*x*F0 | 0*x*11	↔	*x*F1
0*x*F4 ^ 0*x*F5	↔	*x*01

Операция & часто используется для **маскирования** некоторого множества бит. Например, оператор ***w* = *n* & 0177** передает в **w** семь младших бит ***n***, полагая остальные равными нулю

Операции сдвига выполняются также для всех разрядов с потерей выходящих за границы бит

Операция | используется для **включения бит *w* = *x* | *y***, устанавливает в единицу те биты в ***x***, которые равны **1** в ***y***

Необходимо отличать **побитовые** операции **&** и **|** от **логических** операций **&&** и **||** , если ***x* = 1**, ***y* = 2**, то ***x* & *y*** равно **нулю**, а ***x* && *y*** равно **1**.

Унарная операция ~ дает дополнение к целому, т.е. каждый бит со значением **1** получает значение **0** и наоборот

Побитовые логические операции

Если ***выражение_1*** имеет тип ***unsigned***, то при сдвиге вправо освобожда ющиеся разряды гарантированно **заполняются нулями** (логический сдвиг). Выражения типа ***signed*** **могут**, но необязательно, сдвигаться вправо с копированием знакового разряда (арифметический сдвиг). При сдвиге влево освобождающиеся разряды всегда заполняются нулями. Если ***выражение_2*** отрицательно либо больше длины ***выражения_1*** в битах, то результат операции сдвига не определен

Операция

0x81 << 1 ↔ 0x02
0x81 >> 1 ↔ 0x40

Операции сдвига вправо на **k** разрядов весьма эффективны для **деления**, а **сдвиг влево** – для **умножения** целых чисел на **2** в степени **k**:

$x << 1 \leftrightarrow x*2;$
$x >> 1 \leftrightarrow x/2;$
$x << 3 \leftrightarrow x*8.$

Операции сдвига **<<** и **>>** применяются к целочисленным операндам и осуществляют соответственно сдвиг вправо (влево) своего левого операнда на число позиций, задаваемых правым операндом, например, ***x*** **<< 2** сдвигает ***x*** влево на **две позиции**, заполняя освобождающиеся биты нулями (**эквивалентно умножению на 4**)

Побитовые логические операции

Если ***выражение_1*** имеет тип ***unsigned***, то при сдвиге вправо освобожда ющиеся разряды гарантированно **заполняются нулями** (логический сдвиг). Выражения типа ***signed*** **могут**, но необязательно, сдвигаться вправо с копированием знакового разряда (арифметический сдвиг). При сдвиге влево освобождающиеся разряды всегда заполняются нулями. Если ***выражение_2*** отрицательно либо больше длины ***выражения_1*** в битах, то результат операции сдвига не определен

Операция

0x81 << 1 ↔ 0x02
0x81 >> 1 ↔ 0x40

Операции сдвига вправо на **k** разрядов весьма эффективны для **деления**, а **сдвиг влево** – для **умножения** целых чисел на **2** в степени **k**:

x << 1 ↔ *x**2;
x >> 1 ↔ *x*/2 ;

Алгоритм определения четности числа

Операции сдвига << и >> применяются к целочисленным операндам и осуществляют соответственно сдвиг вправо (влево) своего левого операнда на число позиций, задаваемых правым операндом, например, ***x*** **<< 2** сдвигает ***x*** влево на **две позиции**, заполняя освобождающиеся биты нулями (**эквивалентно умножению на 4**)

Побитовые логические операции

Если ***выражение_1*** имеет тип ***unsigned***, то при сдвиге вправо освобождающиеся разряды гарантированно **заполняются нулями** (логический сдвиг). Выражения типа ***signed*** **могут**, но необязательно, сдвигаться вправо с копированием знакового разряда (арифметический сдвиг). При сдвиге влево освобождающиеся разряды всегда заполняются нулями. Если ***выражение_2*** отрицательно либо больше длины ***выражения_1*** в битах, то результат операции сдвига не определен

Операция

0x81 << 1 ↔ 0x02
0x81 >> 1 ↔ 0x40

Операции сдвига вправо на **k** разрядов весьма эффективны для **деления**, а **сдвиг влево** – для **умножения** целых чисел на **2** в степени **k**:

x << 1 ↔ *x**2;
x >> 1 ↔ *x*/2 ;

```
int i=15;
if ( i & 1) printf (" Значение i четно!");
```

Операции сдвига << и >> применяются к целочисленным операндам и осуществляют соответственно сдвиг вправо (влево) своего левого операнда на число позиций, задаваемых правым операндом, например, ***x*** **<< 2** сдвигает ***x*** влево на **две позиции**, заполняя освобождающиеся биты нулями (**эквивалентно умножению на 4**)

Операция «,» (запятая)

Данная операция используется при организации строго гарантированной последовательности вычисления выражений (обычно используется там, где по синтаксису допустима только одна операция, а необходимо разместить две и более, например, в операторе ***for***)

Форма

выражение_1, …, выражение_N;

Выражения **1, 2,…, *N*** вычисляются последовательно друг за другом и результатом операции становится значение последнего выражения ***N***, например:

***m* = (*i* = 1, *j* = *i* ++, *k* = 6, *n* = *i* + *j* + *k*);**

Получим последовательность вычислений:
***i* =1, *j* = *i* =1, *i* = 2, *k* = 6, *n* = 2+1+6,** и в результате ***m* = *n* = 9**