Операции и выражения

Терминология:

- **операция** – действие по преобразованию данных или изменению доступа к ним
- **операнды** – данные, участники операции, результат операции
- **выражение –** линейная последовательность операций и участвующих в них операндов
- **приоритет выполнения –** в выражении последовательность выполнения операций определяется их приоритетами, «приоритетные» скобки меняют последовательность исполнения
- **направление выполнения** (особенность Си) – в последовательности операций *одного приоритета* они выполняются *слева направо* (большинство) или *справа налево*

Пример: a = b = c выполняется как b=c, a=b, т.е. справа налево

Особенности Си

Исторические особенности :

• *архитектурно-ориентированные* – большинство операций соответствуют системе команд «среднестатистического» процессора и выполняются в соответствии с правилами, принятыми в архитектуре. Целый тип = машинное слово

• *многообразие операций* – к операциям относятся элементы синтаксиса, которые в других языках относятся к операторам и пр.:

- традиционные операции: арифметические, сравнение, логические, поразрядные
- **присваивание**
- работа с производными типами данных – доступ к составляющим типам данных - &,*-работа с указателями,[]-доступ к элементу массива,()-вызов функции, «.» (точка) – доступ к элементу структуры
- преобразование типов данных
- условные вычисления в выражении
- перечисление выражений («запятая»)

•*совместимость операций по типам:* – операнды и результат – целые (МС), это позволяет использовать любые сочетания операций ((a>b)+6)

Замечание: ввод/вывод – не операторы, а вызов библиотечных функций (Си) или переопределение операций языка (Си++)

Классификация операций

По приоритетам

Приоритеты (ранги) операций в Си++

Ранг	Операции	Ассоциа-тивность
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)	→

Вывод:лишние () не помешают, если не помните приоритеты

Классификация операций

По назначению :

- арифметические **(+,-,*,/,%)** [cprog 4.2]
- логические **(&&, ||, !)**
- сравнения **(<,>,>=,<=,==,!=)**
- поразрядные (машинно-ориентированные) **(&,|,^,~,<<,>>)** [cprog 9.1]
- присваивание **(=,++,--,+=,-=,*-,/= и т.п.)**
- работа с указателями и памятью **(*,&,sizeof)** [cprog 5.2, 9.2]
- переход к составляющему типу данных **((),*,[], . , ->)** [cprog 5.4]
- явное преобразование типа **((тип))**
- последовательность выражений **(","-запятая)**, условная **(?:)**

Арифметические операции

Операция **%** вычисляет остаток от деления первого операнда на второй. Содержательный смысл: второй операнд-константа выступает ограничителем возможных изменений первого операнда и называется модулем. Название такой операции звучит как **"... по модулю ..."**:

```
a = (a + 1) % 16;          // a присвоить a+1 по модулю 16  0 1 … 14 15 0 1…
a = (++a )% 16;
```

Грабли при вычислениях:

•преобразование типа данных выполняется *для каждой операции отдельно* по общим правилам, независимо от типов последующих операндов

```
double  dd; int a=5; dd=a/4;      //  int/int double=int  dd=2
double  dd; int a=5; dd=a/4.0;  //  int/double double=double  dd=2.5
```

• преобразование размерности, знаковое/беззнаковое, целое/вещественное

```
char b, int a=258,c; c=b=a;                    // значение  b=(char)0x102=2 c=2;
```

Операции сравнения, логический тип

В Си отсутствует особый базовый тип данных для представления логических значений **«истина»** и **«ложь»** *(C++ - есть тип bool)***.** Используются тип int. Значение 0 всегда является **"ложью".** Значение 1 -**"истиной".** Такие значения дают операции сравнения и логические операции. Вообще, в широком смысле любое ненулевое значение является истинным. В такой интерпретации проверяются условия в операторах программы. Поэтому можно записать:

```
if (1) { A } else { B }          // Всегда выполнять B
while (1) { ... }                // «Вечный» цикл
if (k) { A } else { B }          // Эквивалентно if(k !=0)
```

Все операции сравнения дают в качестве результата значения 1 или 0. Следовательно, их можно использовать совместно с арифметическими и другими операциями:

```
a = b > c;              // Запомнить результат сравнения
a = (b > c)* 2;         // Принимает значения 0 или 2
```

< - меньше
> - больше

>= - больше или равно
<= - меньше или равно

== - равно
!= - не равно

Логические операции

Логические операции И (&&) , ИЛИ (||) и НЕ (!) едины для всех языков программирования и соответствуют логическим функциям **И, ИЛИ и НЕ** для логических (булевых) переменных.

«Житейский смысл»:

- **И -** результат «истина» , *одновременно оба* «истина»
- **ИЛИ -** результат «истина» , *хотя бы один* «истина»

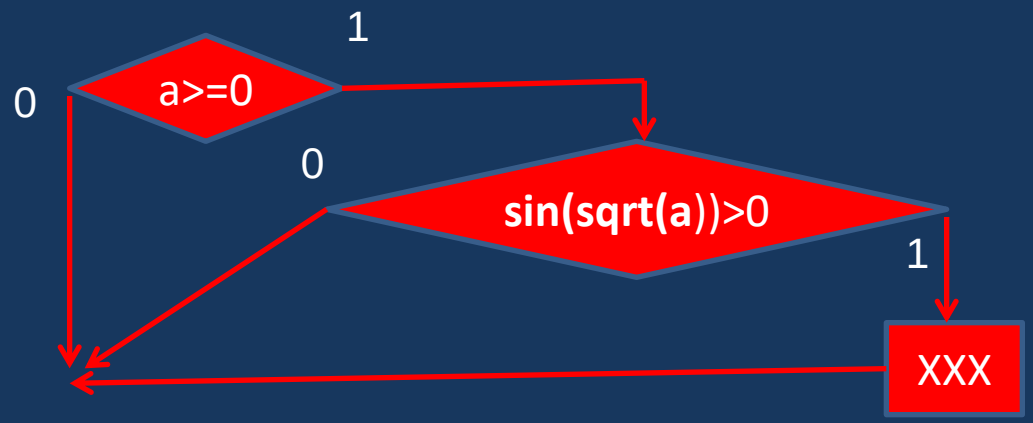
```
if (a < b && b < c)        // если ОДНОВРЕМЕННО ОБА  a < b и b < c, то...
if (a==0 || b > 0)         // если ХОТЯ БЫ ОДИН a==0 или b > 0, то...
```

Особенность Си (реализация в коде):

- если в операции **И** первый операнд «ложь», результат «ложь» и вычисление прекращается
- если в операции **ИЛИ** – первый операнд «истина» - *аналогично*

Пример: второй операнд является корректным только при значении «истина» для первого

```
if (a >=0 && sin(sqrt(a)) >0) { XXX}
```



Логические операции

Логическая инверсия (отрицания) – **«!»**:

- 0 -> 1
- любое не 0 -> 1

```
while(!k) {...}              // эквивалентно while(k==0) {...}
```

Лайфхак: все условия, записанные в заголовках циклов Си-программ, являются условиями продолжения цикла. Если удобнее сформулировать условие завершения, то в заголовке цикла его нужно записать, предварив операцией логической инверсии.

```
// Цикл завершается при обнаружении пары  <0,>0
for (i=1; i<20 && !(A[i-1]<0 && A[i]>0); i++);
```

Полезные эквивалентные преобразования логических выражений:

Инверсия условий, объединенных по И, раскрывается как объединение по ИЛИ обратных условий и наоборот.

```
// Цикл прекращается, когда одновременно оба равны 0
for (i=1; !(A[i-1]==0 && A[i]==0); i++)...

// Цикл продолжается, пока хотя бы один не равен 0
for (i=1; A[i-1]!=0 || A[i]!=0; i++)...
```

&& - логическое И

|| - логическое ИЛИ

! - логическое НЕ (отрицание)

Присваивание

К операциям присваивания относятся все операции, которые меняют значение одного из операндов:

• обычное присваивание **(=);**

• присваивание, соединенное с одной их бинарных операций **(+=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=)** – *расшифровывается a+=b как a=a+b (адресное выражение для **a** вычисляется 1 раз)*

• операции инкремента (++) и декремента (--)

Особенности:

• направление выполнений *справа налево*

• тип выражения в правой части меняется на тип левого операнда (см. приведение)

• результат – значение левой части после присваивания, тип результата – тип переменной левой части

```
int  a,c; double b=2.5;     c=(a=b)+5;              // результат a=b -> значение a типа int
```

Операция присваивания **"="** сохраняет значение выражения, стоящего в левой части, в переменной, а точнее, в адресном выражении, стоящем а правой части. Термин **адресное выражение (l-value)** используется для обозначения тех выражений, которым соответствуют исходные объекты (переменные) в памяти программы. Дуализм понятия «имя переменной»: в левой части - ссылка (адрес этой переменной в памяти), в правой части – ее значение.

Присваивание вида +=

a +=b; // эквивалентно a = a + b;

Эквивалент этой операции верен лишь в первом приближении, потому что в этих операциях левый операнд *(адресное выражение)*, вычисляется один, а не два раза. Например:

A[i++] +=b; // эквивалентно A[i] = A[i] + b; i++;

++/-- Действие и результат

Операция	Действие	Результат
a++	a=a+1	a – до изменения
++a	a=a+1	a+1 – после изменения
a--	a=a-1	a – до изменения
--a	a=a-1	a – после изменения

Результат – то, что используется далее в выражении (значение и тип)
Действие – изменение операндов

Поразрядные операции

Логические операции над каждой парой одноименных разрядов МС [cprog 9.1]

- "|" -поразрядная операция **ИЛИ**
- "&" -поразрядная операция **И**;
- "^" -поразрядная операция **исключающее ИЛИ**;
- "~" -поразрядная операция **инверсия**;
- ">>" -операция **сдвига вправо**;
- "<<" -операция **сдвига влево**.

Явное преобразование типа

Изменение типа операнда к требуемому : (тип)

```
double x,d;          // double x,d; int n;
d = x - (int)x;      // n = x; d = x - d;
                     // дробная часть = исходное double – целая часть
```

Замечание: при присваивании преобразование типа происходит автоматически (явное приведение не требуется)

Работа с указателями и памятью

- **sizeof(выражение), sizeof(тип)** - размер результата выражения или типа данных в байтах
- **&** - переход от объекта (переменной) к ее адресу (указателю)
- ***** - переход от указателя к указуемой переменной – *косвенное обращение по указателю, разыменование* [cprog 5.2]

Переход к составляющему типу данных

- операций не изменяют данные
- осуществляют переход от одной формы представления к другой - от производного типа к его составляющей и наоборот [cprog 5.5]
- используются при работе с производными типами данных (*функции, массивы, структуры, указатели*)

Опера-ция	Тип данных	Операнд	Результат	Действие
[]	Массив	Массив, указатель	Элемент массива	Извлечение (доступ) к элементу массива по индексу
()	Функция	Функция	Результат функции	Вызов функции
*	Указатель	Указатель	Любой	Переход от указателя на объект к объекту
&	Указатель	Любой	Указатель (адрес)	Получение указателя (адреса) объекта (переменной)
.	Структура	Структура (объект)	Элемент структуры	Переход от структуры к ее элементу по имени поля
->	Структура	Указатель на структуру	Элемент структуры	Эквивалент операций * и . (*p).a

Операции «на лету»

Действия, не прерывающие вычисление выражений

• присваивание «на лету»

```
while ((c=getchar()) !='\n') {...c...}        // Символы из потока до конца строки
```

• условная операция – вычисление выражения в зависимости от условия

```
int  a,c;  double b;
c = a > b ? a : b;      // Условие ? Выражение для «истина» : Выражение для «ложь»
```

Замечание: тип выражения выводится статически, в данном случае

```
a > b ? (double)a : b -> double
```

• запятая – перечисление выражений, результат – значение и тип последнего (если используется из списка)

```
for (i=0 , j=n-1; i<j;  i++ , j--)              // Цикл с двумя индексами
      {...A[i]...A[j]...}
```

Приведение типов операндов

ПривЕдение - преобразование от одного вида к другому
ПривИдение – это другое…

Когда происходят (элементы синтаксиса):

- **Присваивание -** значение выражения из правой части запоминается в переменной в левой части

  int a; double b=6.5; a = b*2.2;
- **операция явного преобразования типа** – преобразование в той части выражения, где это требуется

  double b=6.5; int a=5; a = (int)b * 2;
- **при выполнении бинарных операций над операндами различных типов** - более «длинный» операнд превалирует над более «коротким», вещественное - над целым, а беззнаковое над знаковым.

В чем заключается:

- преобразование целой переменной в переменную вещественную (с плавающей точкой) и наоборот
- увеличение или уменьшение разрядности машинного слова, то есть «усечение» или «растягивание» целой переменной
- преобразование знаковой формы представления целого в беззнаковую и наоборот

Приведение типов операндов

Что происходит и какие **ошибки**:

- преобразование целой переменной в переменную вещественную и наоборот – *алгоритм (аппаратный или программный)* преобразования в другой формат.
  Ошибка: переполнение при double -> int
  double dd=1.25E30; int a=dd; // переполнение

- увеличение или уменьшение разрядности машинного слова. Ошибка: потеря значащих разрядов при «укорочении», искажение результата
  char c=514; // c=0x202; реально c=0x02;

Замечание: увеличение размерности сохраняет значение *в той форме,* в которой определен операнд

- увеличение без знака (логическое) – заполняется 0
- увеличение со знаком (арифметическое) – заполняется старшим (знаковым) разрядом

```
7   int main(){
8       char c = -1;
9       short cc = c;
10      int ccc = cc;
11  }
```

Имя	Значение	Тип
c	0xff 'я'	char
cc	0xffff	short
ccc	0xffffffff	int

```
unsigned char c = -1;
printf("c=%d\n", c);
```

```
c=255
```

- преобразование знаковой формы представления целого в беззнаковую и наоборот – *сохранение содержимого МС при изменении формата*
  - unsigned char c=-1; // реально c=255;

Приведение типов операндов

Правила для выражений:

- перед началом выполнения все короткие типы удлиняются до стандартных char,short ->int, float->double
- double + любой -> double + приводится к double
- long + любой -> приводится к long
- unsigned + любой -> приводится к unsigned
- иначе int + int

  Проще:, вещественное превалирует над целым, более «длинный» операнд над более «коротким», беззнаковое над знаковым

«Грабли» при вычислениях

Преобразование типа данных выполняется *для каждой операции отдельно* по общим правилам, независимо от типов последующих операндов

```
double  d1; int a = 5; d1 = a / 4;   //  int/int double=int  dd=2
double  d2; d2 = a / 4.0;            //  int/double double=double  dd=2.5
```

```
d1=1.000000 d2=1.250000
```

Преобразование размерности, потеря значащих разрядов

```
char b;  int aa = 258, c; c = b = aa;   // значение  b=(char)0x102=2 c=2;
printf("aa=%d c=%d\n", aa, c);
```

```
aa=258 c=2
```

Диапазон знаковый/беззнаковый char – код кириллицы <0

```
char cc = getchar();                // Диапазон -127...+127
unsigned char uc = getchar();       // Диапазон 0...255
printf("cc=%d uc=%d\n", cc, uc);
```

```
яя
cc=-17 uc=239
```

Совместимость по результату, компилируется *с предупреждением*

```
int main(){
    setlocale(LC_ALL, "Russian");   // Установка преобразования при выводе
    int uu = 6;
    if (uu = 5)     // Присваивание вместо сравнения (uu=5)!=0 - истина
        printf("uu равно 5");
    else
        printf("uu не равно 5");
```

```
uu равно 5
```

Вывод: не мешайте без необходимости разные типы в одном выражении

«Грабли» при вычислениях

Совместимость по результату, компилируется *с предупреждением*

```
uu меньше 4
```

```c
int main(){
    setlocale(LC_ALL, "Russian");   // Установка преобразования при выводе
    int uu = 6;
    if (uu << 4)   // Сдвиг вместо сравнения uu << 4 - 0x60 !=0 - истина
        printf("uu меньше 4");
    else
        printf("uu больше или равно 4");
```

Вывод: следите за синтаксисом операций

Контрольные вопросы

Определить результат вычисления выражений в удобной для этого форме (целое со знаком или без, вещественное, маш.слово в 16СС).
Объясните особенности используемого синтаксиса.
Какие особенности Си определяют результат?

1. int a=257; char c=a;
2. int a=1, b=0; if (a) b++;
3. int a=5; double b; b=a/2;
4. int a=125; double b; b=a/10.0;
5. int a=125; double b; b=((double)a)/10;
6. char c; if((c=getchar())!=' ') putchar(c);
7. int i1=0xFFFFFFFF; i1++; // sizeof(int)==4
8. unsigned u2=-1,u1=1, u=0; if (u1>u2) u++;
9. int i1=0x01FF; char c; c = i1; i1 = c;
10. int i1 = 0x01FF; unsigned char c; c = i1; i1 = c;
11. double d1,d2; d1=2.56; d2=(int)d1 + 1.5;
12. double d1,d2; d2=(int)(d1 + 1.5);
13. double d1=2.56; int i = (d1 - (int)d1) * 10;
14. i=0; if (i++) i++;
15. i=0; if (++i) i++;

Контрольные вопросы

16. int b,a; if (a!=0 && b/a==10)... // Почему в Basic нужно писать через два if
17. double a; if (a <0 || sqrt(a)>10)... // Почему в Basic нужно писать через два if
18. int a=125,b,c; c=(b=a/10)/10;
19. int a=125,b,c; c=b=a/10;
20. int double a=12.5,c; int b; c=b=a;
21. int a,b,c; c = b==0 ? MAXINT : a/b;
22. int a; double b; a=(int)b + (b-(int)b) >0.5 ? 1 : 0);
23. int a=5,b=7,c=4,d; d = (a>b)+(b>c)+(a>c);
24. int a=0x1F; int b = !a; int c=~a;