C Programming: Arithmetic and Logic Operations

A. Sahu and P. Mitra

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Algebra: BEDMAS/PEDMAS Rule (recap)

- B-E-DM-AS or P-E-DM-AS
- B/P : Bracket or Parenthesis ( )
  - In C, only ( ) used for expression
  - Curly braces {}, and square bracket [] used for some other purpose.
    - Again [] may involves in expression as in the form of array access
- E : Exponentiation
- DM: Division and Multiplication
- AS : Addition and Subtraction

BEDMAS equivalent in C
Arithmetic Operators Precedence Rule
(recap)

Operator(s)	Precedence & Associativity
( )	Evaluated first. If **nested (embedded)**, innermost first.
* / %	Evaluated second. If there are several, evaluated left to right.
+ -	Evaluated third. If there are several, evaluated left to right.
=	Evaluated last, right to left.

Using Parentheses

- Use parentheses to change the order in which an expression is evaluated.

  `a + b * c` Would multiply b * c first, then add a to the result.

  If you really want the sum of a and b to be multiplied by c, use parentheses to force the evaluation to be done in the order you want.

  `(a + b) * c`

- Also use parentheses to clarify a complex expression.

Practice With Evaluating Expressions (recap)

Given integer variables a, b, c, d, and e, where a = 1, b = 2, c = 3, d = 4,

evaluate the following expressions:

```
a + b - c + d
a * b / c
1 + a * b % c
a + d % b - c
e = b = d + c / b - a
```

Practice With Evaluating Expressions (recap)

Given integer variables a, b, c, d, and e, where a = 1, b = 2, c = 3, d = 4,

evaluate the following expressions:

```
a + b - c + d =3-3+4=0+4=4
a * b / c = 2/3+4=0+4=4
1 + a * b % c =1+2%3=1+2=3
a + d % b - c =1+0-3=1-3=-2
e = b = d + c / b – a
```

Increment and Decrement Operators

- The **increment operator** ++
- The **decrement operator** --
- Precedence: lower than (), but higher than * / and %
- Associativity: right to left
- Increment and decrement operators can only be applied to variables, not to constants or expressions

Increment Operator

- If we want to add one to a variable, we can say:

```
count = count + 1 ;
```

- Programs often contain statements that increment variables, so to save on typing, C provides these shortcuts:

```
count++ ;  OR  ++count ;
```

Both do the same thing. They change the value of count by adding one to it.

Postincrement Operator

- The position of the ++ determines when the value is incremented. If the ++ is after the variable, then the incrementing is done last (a **postincrement**).

```
int amount, count ;

count = 3 ;
amount = 2 * count++ ;
```

- amount gets the value of 2 * 3, which is 6, and then 1 gets added to count.
- So, after executing the last line, amount is 6 and count is 4.

Preincrement Operator

- If the ++ is before the variable, then the incrementing is done first (a **preincrement**).

```
int amount, count ;

count = 3 ;
amount = 2 * ++count ;
```

- 1 gets added to count first, then amount gets the value of 2 * 4, which is 8.
- So, after executing the last line, amount is 8 and count is 4.

A Hand Trace Example

```
int  ans, val= 4 ;
```

Code	Val	Ans
	4	garbage

```
val = val + 1 ;
val++ ;
++val ;
ans = 2 * val++ ;
ans = ++val / 2 ;
val-- ;
--val ;
ans = --val * 2 ;
ans = val-- / 3 ;
```

A Hand Trace Example

```
int  ans, val= 4 ;
```

Code	Val	Ans
	4	garbage
`val = val + 1 ;`	5	
`val++ ;`	6	
`++val ;`	7	
`ans = 2 * val++ ;`	8	14
`ans = ++val / 2 ;`	9	4
`val-- ;`	8	
`--val ;`	7	
`ans = --val * 2 ;`	6	12
`ans = val-- / 3 ;`	5	2

C Code : Previous Example

```
int main(){
 int ans, val=4;
 val = val + 1 ;
 printf("ans=%d val=%d\n",ans,val);
 val++ ; ++val ;
 printf("ans=%d val=%d\n",ans,val);
 ans = 2 * val++ ;
 printf("ans=%d val=%d\n",ans,val);
 ans=++val/2; val--;--val;
 printf("ans=%d val=%d\n",ans,val);
 ans=--val*2;
 printf("ans=%d val=%d\n",ans,val);
 ans = val-- / 3 ;
 printf("ans=%d val=%d\n",ans,val);
 return 0;
 }
```

Practice

Given

```
int a = 1, b = 2, c = 3 ;
```

What is the value of this expression?

```
++a * b - c--
```

What are the new values of a, b, and c?

Practice

Given

```
int a = 1, b = 2, c = 3 ;
```

What is the value of this expression?

```
++a * b - c--
```

=1

What are the new values of a, b, and c?

a =2 c=2

More Practice

Given

```
int a =1, b =2, c =3, d =4 ;
```

What is the value of this expression?

```
++b / c + a * d++
```

What are the new values of a, b, c, and d?

More Practice

Given

```
int a =1, b =2, c =3, d =4 ;
```

What is the value of this expression?

```
++b / c + a * d++
```

= 1+4 =5

What are the new values of a, b, c, and d?

a=1, b=3,c=3,d=5

Assignment Operators

= += -= *= /= %=

Statement	Equivalent Statement
a = a+2 ;	a += 2 ;
a = a-3 ;	a -= 3 ;
a = a*2 ;	a *= 2 ;
a = a/4 ;	a /= 4 ;
a = a%2 ;	a %= 2 ;
b = b+(c+2);	b += c + 2 ;
d =d*(e-5);	d *= e - 5 ;

Practice with Assignment Operators

```
int i = 1, j = 2, k = 3, m = 4 ;
```

Expression	Value
`i += j + k`	
`j *= k = m + 5`	
`k -= m /= j * 2`	

Practice with Assignment Operators

```
int i = 1, j = 2, k = 3, m = 4 ;
```

Expression	Value
`i += j + k`	**i=6**
`j *= k = m + 5`	**k=9, j=18**
`k -= m /= j * 2`	**m=1, k=2**

Relational Expressions and Evaluation

Relational Operators

Operator	Meaning
<	Less than
>	Greater than
<=	less than or equal to
>=	greater than or equal to
==	is equal to
!=	is not equal to

Relational expressions evaluate to the integer values 1 (true) or 0 (false).

All of these operators are called **binary operators** because they take two expressions as **operands**.

Practice with Relational Expressions

```
int a = 1, b = 2, c = 3 ;
```

Expression	Value	Expression	Value
`a < c`		`(a + b)>= c`	
`b <= c`		`(a + b)== c`	
`c <= a`		`a != b`	
`a > b`		`(a + b)!= c`	
`b >= c`			

Practice with Relational Expressions

```
int a = 1, b = 2, c = 3 ;
```

Expression	Value	Expression	Value
`a < c`	T	`(a + b)>= c`	T
`b <= c`	T	`(a + b)== c`	T
`c <= a`	F	`a != b`	T
`a > b`	F	`(a + b)!= c`	F
`b >= c`	F		

Arithmetic Expressions: True or False

- **Arithmetic expressions** evaluate to numeric values.
- An arithmetic expression that has a **value of zero is false**.
- An arithmetic expression that has a value **other than zero is true**.

Practice with Arithmetic Expressions

```
int       a = 1, b = 2, c = 3 ;
float   x = 3.33, y = 6.66 ;
```

Expression	Numeric Value	True/False
a + b		
b-2*a		
c– b–a		
c–a		
y–x		
y-2*x		

Practice with Arithmetic Expressions

```
int       a = 1, b = 2, c = 3 ;
float   x = 3.33, y = 6.66 ;
```

Expression	Numeric Value	True/False
a + b	3	T
b-2*a	0	F
c– b–a	0	F
c–a	2	T
y–x	3.33	T
y-2*x	0.0	F

Structured Programming

- All programs can be written in terms of only three control structures
  - **Sequence, selection and repetition**
- The **sequence** structure
  - Unless otherwise directed, the statements are executed in the order in which they are written.
- The **selection** structure
  - Used to choose among alternative courses of action.
- The **repetition** structure
  - Allows an action to be repeated while some condition remains true.

Selection: the if statement

```
if  ( condition )
{
 statement(s) /* body of the if
                 statement */

}
```

The braces are not required if the body contains only a single statement.

However, they are a good idea and are required by the C Coding Standards.

Examples

```
if ( age >= 18 )
{
    printf(“Vote!\n”) ;
}
if ( value == 0 )
{
  printf(“You entered a zero.\n”) ;
  printf (“Please try again.\n”) ;
}
```

Good Programming Practice

- Always place braces around the body of an if statement.
- Advantages:
  - Easier to read
  - Will not forget to add the braces if you go back and add a second statement to the body
  - Less likely to make a semantic error
- **Indent the body of the if statement 3 to 4 spaces -- be consistent!**

Selection: the if-else statement

```
if  ( condition )
{
     statement(s)/*if clause */
}
else
{
     statement(s)/*else clause */
}
```

Example

```
if ( age >= 18 )
{
    printf(“Vote!\n”) ;
}
else
{
    printf(“Maybe next time!\n”) ;
}
```

Good Programming Practice

- Always place braces around the bodies of the if and else clauses of an if-else statement.
- Advantages:
  - Easier to read
  - Will not forget to add the braces if you go back and add a second statement to the clause
  - Less likely to make a semantic error
- Indent the bodies of the if and else clauses 3 to 5 spaces -- be consistent!

Nesting of if-else Statements

```
if ( condition1 )
{
     statement(s)
}
else if ( condition2 )
{
    statement(s)
}
. . . /* more else clauses may be here */
else
{
    statement(s)  /* the default case */
}
```

Good Example : 2 if 2 else

```
if ( value == 0 )
{
   printf("Value you entered was 0\n");
}
else if ( value < 0 )
{
   printf("%d is negative.\n", value) ;
}
else
{
   printf("%d is positive.\n", value) ;
}
```

Bad Example : 2 if 1 else

```
if ( n > 0 )
    if ( a > b  )
        z=a;
else
     z=b;
```

```
if ( n > 0 )
  if ( a > b )
     z=a;
  else
     z=b;
```

```
if ( n > 0 )
{
  if (a> b)
     z=a;
}
else
     z=b;
```

Indentation will not ensure result :

else match with closest if

Code of Red box behaves like Code of Green box

Gotcha! = versus ==

```
int    a = 2 ;
if ( a = 1 )/* semantic(logic) Err! */
{
    printf (“a is one\n”) ;
}
else if (a == 2 )
{
    printf (“a is two\n”) ;
}  else {
    printf (“a is %d\n”, a) ;
}
```

Gotcha ..

- The statement if (a = 1) is syntactically correct, so no error message will be produced. (Some compilers will produce a warning.) However, a semantic (logic) error will occur.
- An assignment expression has a value -- the value being assigned. In this case the value being assigned is 1, which is true.
- If the value being assigned was 0, then the expression would evaluate to 0, which is false.
- This is a VERY common error. So, if your if-else structure always executes the same, look for this typographical error.

Logical Operators

- So far we have seen only **simple conditions**.

```
if ( count > 10 ) . . .
```

- Sometimes we need to test multiple conditions in order to make a decision.
- **Logical operators** are used for combining simple conditions to make **complex conditions**.

&&	is AND	if (x > 5 && y < 6)
||	is OR	if (z == 0 || x > 10)
!	is NOT	if (! (bob > 42))

Example Use of &&

```
if ( age < 1 && gender == ‘m’)
{
    printf (“Infant boy\n”) ;
}
```

Truth Table for &&

$Expression_1$	$Expression_2$	$Expression_1$ && $Expression_2$
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Exp_1 && Exp_2 && … && Exp_n will evaluate to 1 (true) only if ALL **subconditions** are true.

Example Use of ||

```
if (grade==‘E’ || grade==‘F’)
{
 printf(“See you next semester!\n”) ;
}
```

Truth Table for ||

$Expression_1$	$Expression_2$	$Expression_1$ || $Expression_2$
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Exp_1 && Exp_2 && … && Exp_n will evaluate to 1 (true) if only ONE subcondition is true.

Example Use of !

```
if (!(x==2)) /* Same as (x!=2) */
{
  printf("x is not equal to 2") ;
}
```

Truth Table for !

Expression	! Expression
0	1
nonzero	0

Operator Precedence and Associativity

Precedence	Associativity
( )	left to right/inside-out
++ -- ! + (unary) - (unary) (type)	right to left
* / %	left to right
+ (addition) - (subtraction)	left to right
< <= > >=	left ot right
== !=	left to right
&&	left to right
||	left to right
= += -= *= /= %=	right to left
, (comma)	right to left

Thanks