#5 Arithmetic Instructions

By Saurabh Shukla | 2017©mysirg.com

Arithmetic Instructions

An instruction which is used to manipulate data using operators is known as arithmetic instruction.

Operator requires operands (data) to perform certain operation. For example 3+4 is an expression with one operator and two operands. Operator is + and 3, 4 are the operands.

On the basis of number of operands required to perform operation by the operator, we can classify them as:

- Unary operators
- Binary Operators
- Ternary Operators

Unary operators are those which require on one operand, binary operators require two operands, whereas ternary operator needs three operands to perform their tasks.

What matter most is the precedence of operator. When you have an expression with multiple operators in it, you have to rule out which one to solve first. In mathematics, you must know about BODMAS rule. It is simply to remember priority of operators.

Remember: There is no BODMAS rule in C language

Just because there are so many operators in C language, it is not that easy to remember there precedence thus you can categorize operators according to their behaviour as well as priority rule. It will help you to remember priority of operators.

Types of Operators

- Unary Operators
- Arithmetic Operators
- Bitwise Operators
- Relational Operators
- Logical Operators
- Conditional Operator
- Assignment Operators

Unary Operators (+,-,++,--,sizeof)

These operators require only one operand to perform their operation.

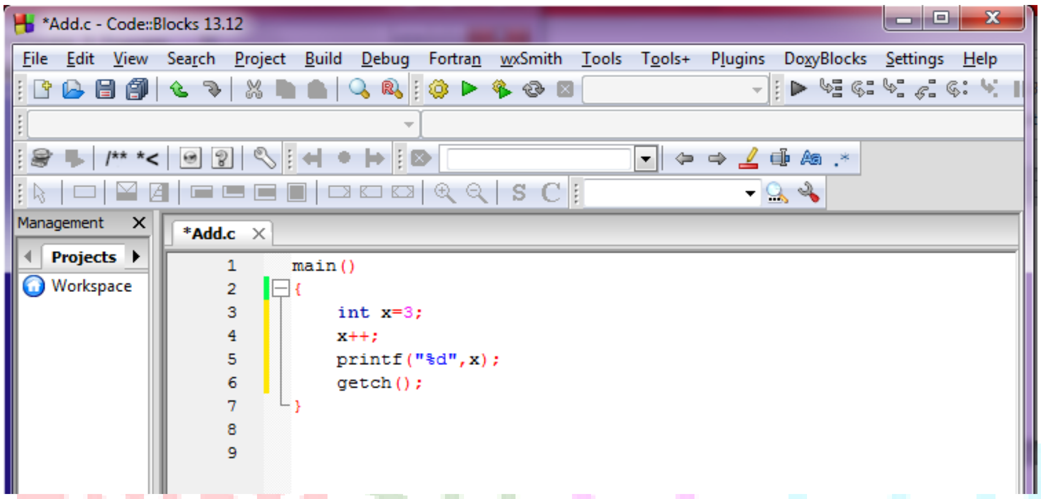
Unary + and unary -

These are not addition or subtraction operators, but they are unary plus and unary minus, used to indicate sign of a number.

For example: -3, +5

Increment Operator ++

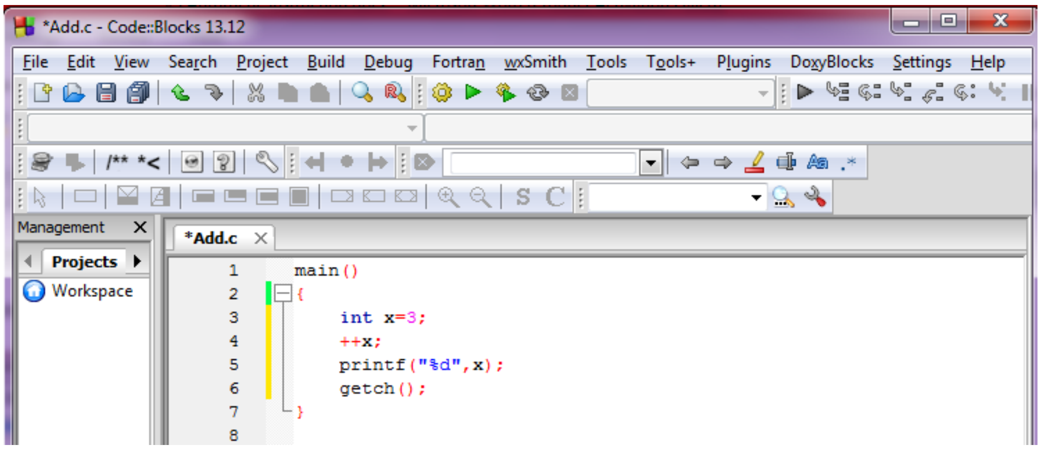
Increment operator increases value of the operand by 1.



```
main()
{
    int x=3;
    x++;
    printf("%d",x);
    getch();
}
```

In the above program x is a variable containing 3 initially. In the next step increment operator is used to increase its value by one. The updated value of x is now 4, so the output of the program is 4.

You can also use operator before operand.

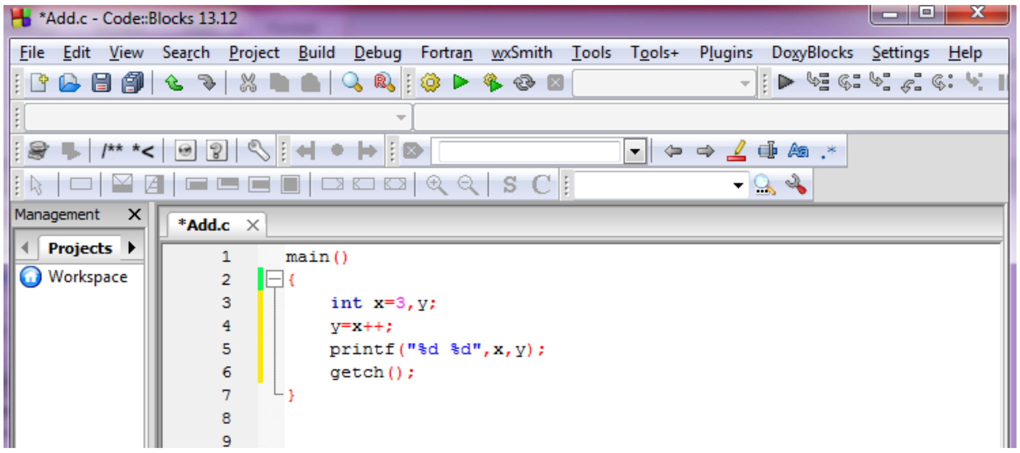


```
main()
{
    int x=3;
    ++x;
    printf("%d",x);
    getch();
}
```

The output of the above program is also 4.

Both the styles of using increment operator (x++ is post increment and ++x is pre increment) have the same function but with different priorities.

Pre increment has high priority among all the operators but post increment has the least priority among all the operators, even less than the assignment operators.

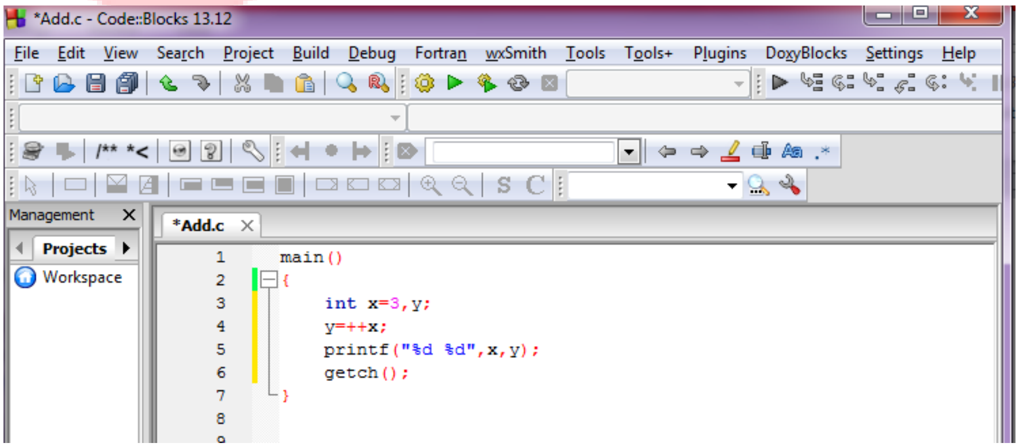


```
main()
{
    int x=3,y;
    y=x++;
    printf("%d %d",x,y);
    getch();
}
```

Try to evaluate the output of above code.

- Initially x contains 3.
- In the expression y=x++; post increment has low priority than assignment operator, thus value of x is copied to y.
- Now x and y both contains 3.
- Increment operator works on x. Value of x becomes 4.
- Final value of x is 4 and y is 3, so the output is 4 3

Now try this program. Find the output.



```
main()
{
    int x=3,y;
    y=++x;
    printf("%d %d",x,y);
    getch();
}
```

- Initially x contains 3.
- In the expression y=++x; pre increment has higher priority than assignment operator, thus value of x is incremented to 4.
- Now x contains 4
- Value of x is copied to y, which is 4 at this moment
- Final value of x is 4 and y is 4, so the output is 4 4

Decrement operator --

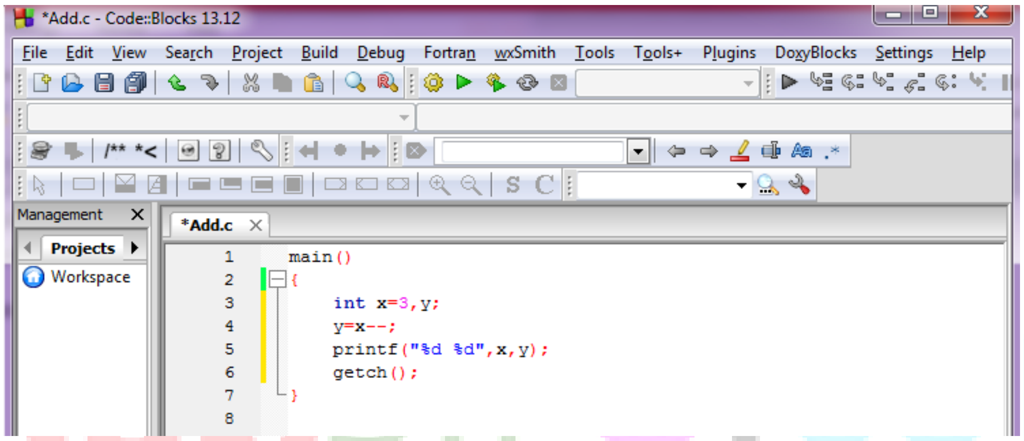
The job of decrement operator is to decrease value of the operand by one. Just like increment operator, decrement operator can be used either before operand or after operand.

x--; post decrement

--x; pre decrement

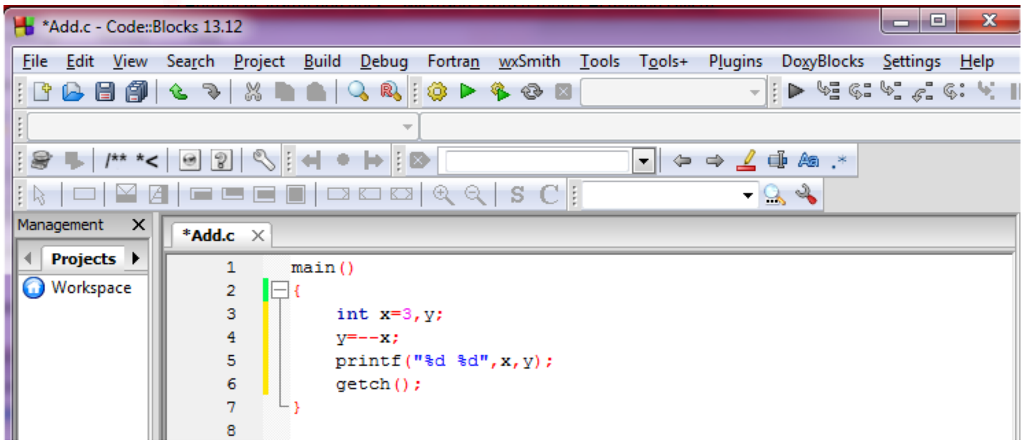
Priority of pre decrement is high among all the operators and priority of post decrement operator is least among all the operators.

Do the following output exercise yourself:



```
main()
{
    int x=3,y;
    y=x--;
    printf("%d %d",x,y);
    getch();
}
```

The output of above code is 2 3



```
main()
{
    int x=3,y;
    y=--x;
    printf("%d %d",x,y);
    getch();
}
```

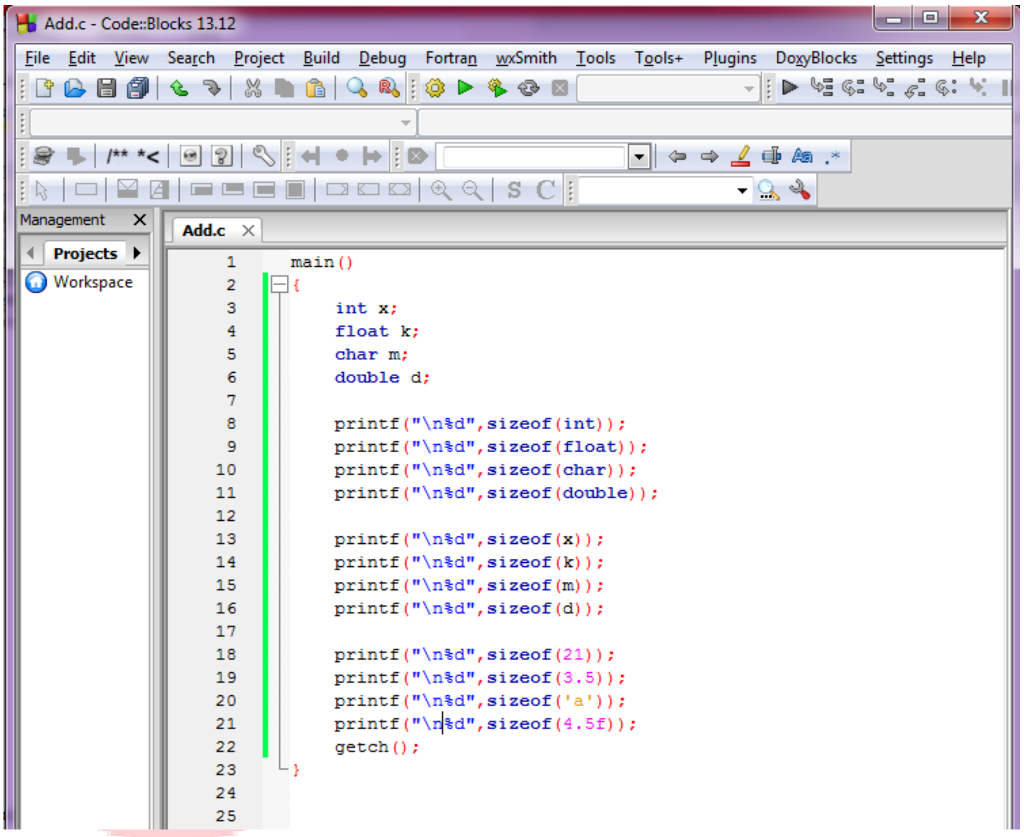
The output of above code is 2 2

Remember: You cannot use constant as an operand of increment or decrement operator. For example 3++; is invalid.

sizeof operator

It is also a keyword. sizeof operator takes one operand as it is unary operator and tells its size in bytes. Operand of sizeof can be a data type, variable or constant.

The below program depicts the behaviour of sizeof operator.


Add.c - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects
Workspace
Add.c

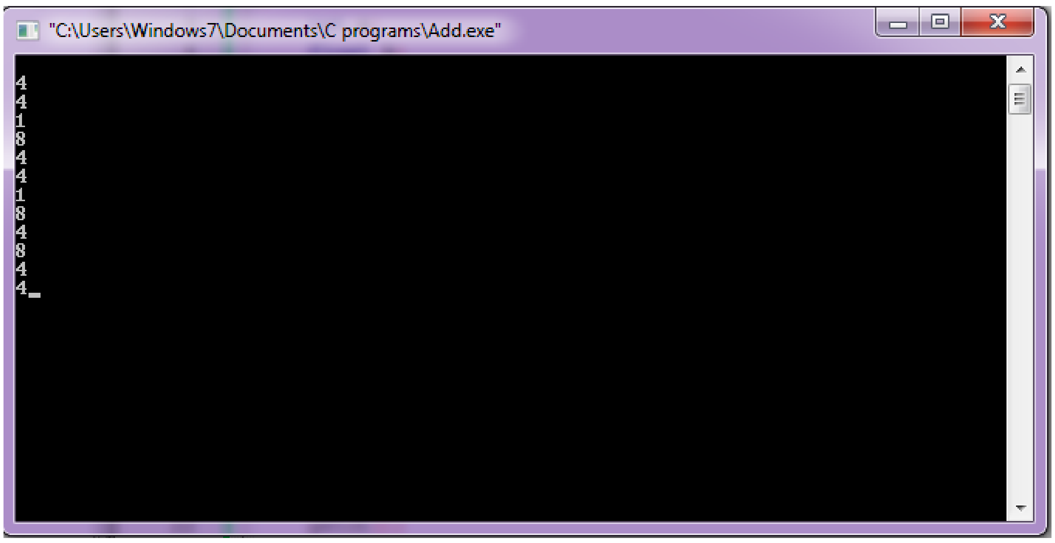
```c
main()
{
    int x;
    float k;
    char m;
    double d;

    printf("\n%d",sizeof(int));
    printf("\n%d",sizeof(float));
    printf("\n%d",sizeof(char));
    printf("\n%d",sizeof(double));

    printf("\n%d",sizeof(x));
    printf("\n%d",sizeof(k));
    printf("\n%d",sizeof(m));
    printf("\n%d",sizeof(d));

    printf("\n%d",sizeof(21));
    printf("\n%d",sizeof(3.5));
    printf("\n%d",sizeof('a'));
    printf("\n%d",sizeof(4.5f));
    getch();
}
```

The output of the program is



```

4
4
1
8
4
4
1
8
4
8
4
4_
```

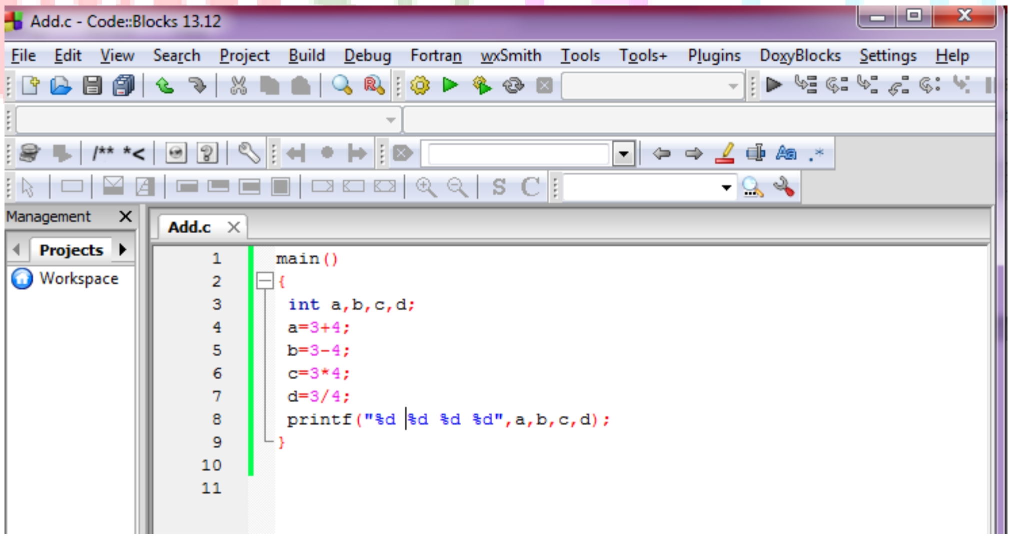
Remember

- int variable in 16 bit compilers (turbo) will consume 2 bytes in memory but in 32 bit or 64 bit compilers (GCC-Code blocks or DEV CPP) it consumes 4 bytes in memory
- char variable take 1 byte in memory but char constant is treated as an integer value (ASCII code), hence assumed as an integer value. sizeof consider it an integer and size is shown 4 bytes
- 3.5 is double and not float
- 4.5f, suffix f tells the compiler that the value is float type

Arithmetic Operators (* , /, %, +, -)

- Priority of *, /, % is higher than +, - operators.
- Unlike mathematics priority of *, / and % are same. Similarly, priority of + and – is same.
- * is used for multiplication
- / is used for division
- % is called modulus operator and used to get remainder
- + is for addition
- - is for subtraction
- If the single expression has more than one arithmetic operators of same priority, they will be evaluated from left to right in the expression. This is called associativity rule.

Find the output of the following program


Add.c - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects
Workspace
Add.c

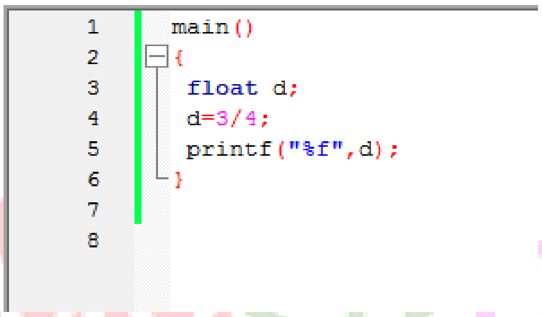
```
main()
{
 int a,b,c,d;
 a=3+4;
 b=3-4;
 c=3*4;
 d=3/4;
 printf("%d %d %d %d",a,b,c,d);
}
```

Output is:

7 -1 12 0

- The value of a is 7 which is the result of 3+4
- The value of b is -1 which is the result of 3-4

- The value of c is 12 which is the result of 3*4
- The value of d is 0 which is the result of 3/4. This may surprise you a bit.
  - In C, numbers are of two types, real and integer.
  - When we operate two integers, result will be integer only
  - So 3/4 is 0 instead of 0.75 (which should be mathematically)
  - Simply discard decimal point and digits after decimal point, only consider the integer part (left of decimal point)
- At least one operand is real, the result will be real
  - The result of 3.0/4 is 0.75
  - The result of 3/4.0 is 0.75
  - The result of 3.0/4.0 is 0.75
- Someone might have this confusion that what if we take variable of type float or double to store the result of 3/4, would it leads to the mathematically correct result.



```c
main()
{
  float d;
  d=3/4;
  printf("%f",d);
}
```

- The output of above program is 0.000000
  - The result contains decimal representation due to %f format specifier
  - Since in the expression d=3/4, divide operator has higher precedence than assignment operator, division performs first. 3 and 4 both are integers therefore outcome can only be integer value. Resulting expression becomes d=0;
- Modulus operator is used to find remainder value and can be worked only on integer operands. So the following expression is invalid in C language.
  - 3.5%2;
- Observe the result of following operations
  - 3%2 => 1
    - Dividing 3 by 2, you will get 1 as a remainder
  - 5%2 => 1
    - Dividing 5 by 2, you will get 1 as a remainder
  - 17%5 => 2
    - Dividing 17 by 5, you will get 2 as a remainder
  - 91%10 =>1
    - Dividing any number by 10 always give last digit as a remainder (You need not to be a genius to understand this concept)
  - 15%5 =>0
    - Perfect division always gives 0 as a remainder. This can be used to test divisibility.

- 3%4 =>3
  - When dividend is smaller than the divisor, the remainder is always the smaller one
- 2%5 =>2
  - 2 is smaller than 5, thus the result is 2

Bitwise Operators (&, |, ^, ~, >>, <<)

Operators that perform on bits (0 or 1) are known as bitwise operators. There are six bitwise operators.

You must know how to convert decimal number into binary and binary number to decimal.

(See reference link at the end of this chapter for the training videos of converting number system)

Behaviour of bitwise operators

Nature of operators

Bitwise AND (&)

0&0 is 0

0&1 is 0

1&0 is 0

1&1 is 1

Bitwise OR (|)

0|0 is 0

0|1 is 1

1|0 is 1

1|1 is 1

Bitwise XOR (^)

0^0 is 0

0^1 is 1

1^0 is 1

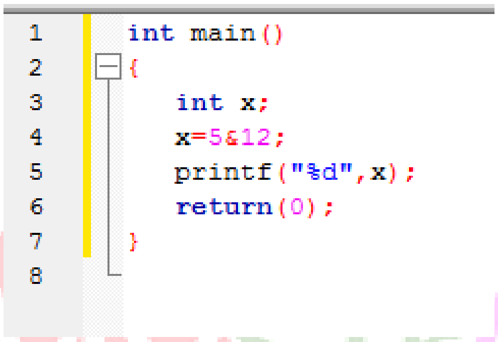
1^1 is 0

Bitwise NOT (~)

~0 is 1

~1 is 0

- To solve bitwise operators &, | and ^, you need to convert both the operands into their binary equivalent. Represent in 32 bits representation. (integer takes 4 bytes in memory)
- Perform operation between every pair of corresponding bits according to the behaviour described above.

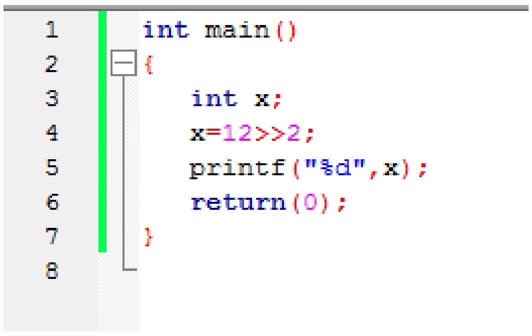


```
int main()
{
    int x;
    x=5&12;
    printf("%d",x);
    return(0);
}
```

- The output is 4
    - Bitwise AND applies on 5 and 12. We need to convert them in binary.

```
5  =00000000 00000000 00000000 00000101
12 =00000000 00000000 00000000 00001100
&  ————————————————------------------------
4  =00000000 00000000 00000000 00000100
```

- In the case of left or right shift operation, you need to convert only left operand into its binary equivalent. For example in the expression 12>>2; you need to convert 12 into its binary and perform right shift two times.

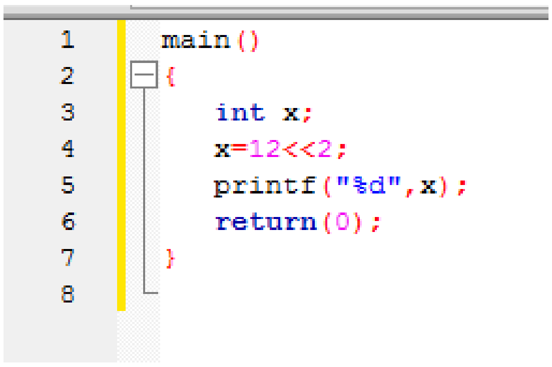


```
int main()
{
    int x;
    x=12>>2;
    printf("%d",x);
    return(0);
}
```

- Convert 12 into binary and shift bits to their right 2 times. This makes last two bits out and two new bits (always 0) append at the left.

12=00000000 00000000 00000000 00001100
Right shift two times
3 =00000000 00000000 00000000 00000011

- In the case of left shift, everything is same except that the shifting of bits is towards left.



```
main()
{
    int x;
    x=12<<2;
    printf("%d",x);
    return(0);
}
```

  - Convert 12 into binary and shift bits to their left 2 times. This makes last two left most bits out and two new bits (always 0) append at the right.
    12=00000000 00000000 00000000 00001100
    Left shift two times
    48=00000000 00000000 00000000 00110000

Relational operators (<, >, <=, >=, ==, !=)

Relational operators are useful in comparison of two values. Operators <(less than), >(greater than), <= (less than or equal to) and >=(greater than or equal to) have higher precedence than the operators ==(equal to) and != (not equal to).

Relational Operators are used to states the truth value of the expression. Result of these operators are either 1 (for true) or 0 (for false).

- 5>4 is evaluated as 1
  - Read the expression, five is greater than four. Truth value of the statement is true, so the result is 1
- 4<4 is evaluated as 0
  - Four less than four, truth value is false
- 3!=4 is evaluated as 1
  - Three is not equal to 4, the truth value of statement is true
- 5>4>3 is evaluated as 0
  - When there are more than one relational operator of same rank, evaluate the operator which comes first from the left in the expression. It is because the associativity rule for relational operators is from left to right.
    5>4>3 (solve 5>4 first)
    1>3 (result of 5>4 is 1)

0 (result of 1>3)

Logical Operators (!, &&, ||)

You often need to combine two statements logically to form a single statement for example there are two statements

1. Marks greater than 50
2. Marks less than 60

On the basis of value of marks, you can comment about the truth value of above two statements. They are independent statements, but what if you need to say something like marks must be between 50 and 60. You want a single statement stating marks greater than 50 and marks lesser than 60. You want to combine both the statements to make it one. Here comes the need of logical operators. You can make a single statement like:

1. Marks greater than 50 and Marks less than 60

Mathematically you can write the same statement as

- Marks>50 && Marks<60 (assume Marks is some variable with value of marks obtained)

Priority of Logical NOT (!) operator is not only just higher than the remaining two logical operator but higher than all other operators too, as it is also a unary operator.

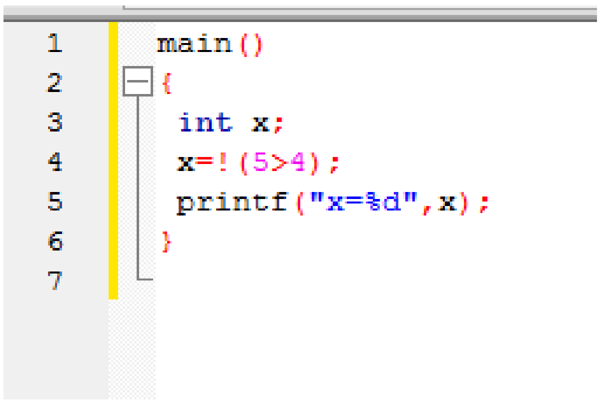
Priority of logical AND (&&) is higher than logical OR (||).

Logical NOT (!)

The behaviour of Logical NOT operator is to alter the truth value of the statement.

! TRUE =FALSE

! FALSE =TRUE



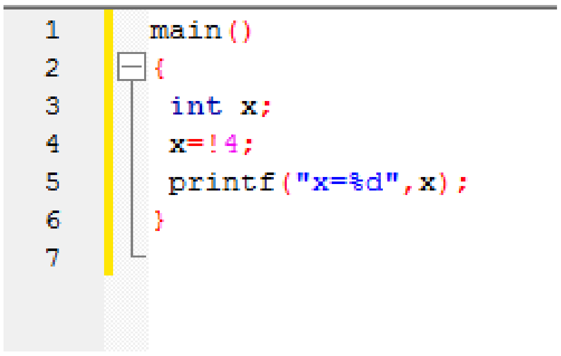
```
main()
{
 int x;
 x=!(5>4);
 printf("x=%d",x);
}
```

The output of the above program is

```
x=0
```

In the expression x=!(5>4), bracket operates first, so the result of 5>4 is 1 (true). This result becomes the operand of logical NOT operator, which makes it 0 (false). Thus the value stored in x is 0.

Try to find the output of the next program



```
main()
{
 int x;
 x=!4;
 printf("x=%d",x);
}
```

The output of above program is

```
x=0
```

In the expression x=!4, 4 is the operand of logical NOT operator. 4 is treated as true. Remember every non-zero value is true and zero is false. 4 is a non-zero value, so it is treated as true. Now logical NOT operator inverts the truth value from true to false and therefore 0 is stored in x.

Logical AND operator (&&)

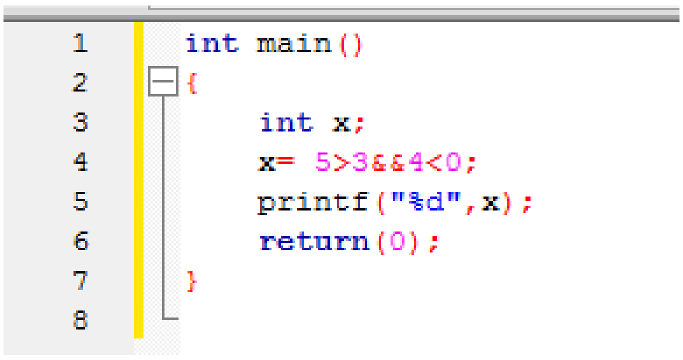
Logical AND operator is used to combine two expression, thus it is a binary operator. The format of expression using && operator is :

- Expression1 && Expression2

The behaviour of logical AND is describes as:

Expression 1	Expression2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Find the output of following code



```
int main()
{
    int x;
    x= 5>3&&4<0;
    printf("%d",x);
    return(0);
}
```

Output is

```
0
```

Here, two conditions 5>3 and 4<0 are combined to form a single condition using && operator as 5>3&&4<0. Condition one is evaluated as TRUE as 5 is greater than 3. Since condition one is TRUE condition two is tested and evaluated as FALSE as 4 is not less than 0. According to the above chart, T&&F is treated as FALSE thus 0 is stored in x.

Logical OR (||)

Logical OR operator is also used to combine two expressions. This is a binary operator.

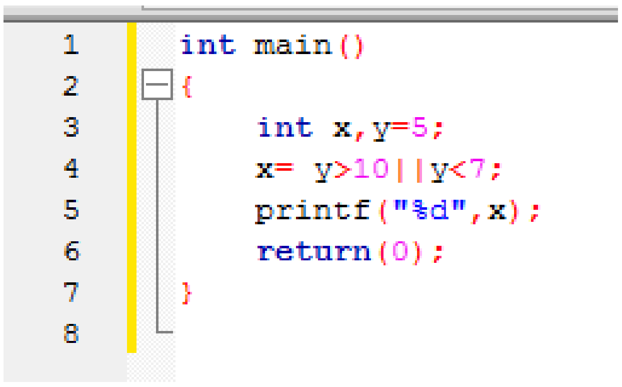
The format of expression using || operator is :

- Expression1 || Expression2

The behaviour of logical OR operator is as follows

Expression 1	Expression2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Find the output of the following program



```
int main()
{
    int x,y=5;
    x= y>10||y<7;
    printf("%d",x);
    return(0);
}
```

The output is

```
1
```

In the expression x=y>10 ||y<7, y>10 is false, so we check expression 2 and y<7 is true, therefore operands of logical OR operator are false and true. The result is then evaluated as true, this result is assigned to variable x. Since true is represented as 1 (one) therefore x is assigned 1.

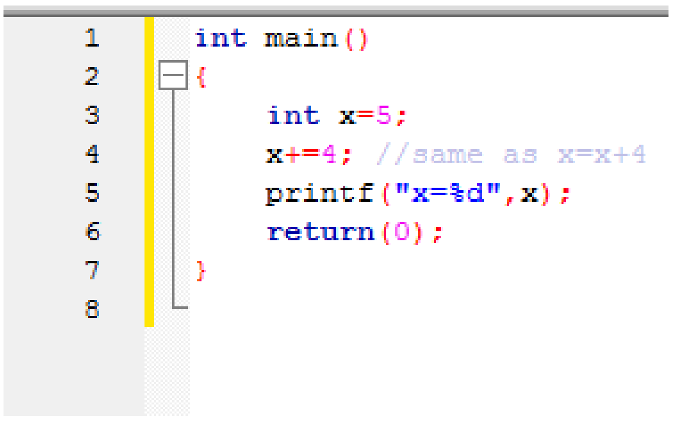
Assignment Operator (= ,+=, -=, *=, /=, %=, &=,|=, ^=)

Assignment operator is the most used operator in the expression. It is sometimes misinterpreted as equal to operators by beginners. Assignment operator is used to assign value to the variable.

It is a binary operator where left operand must be a variable.

- x=4;
  - In this expression value 4 is assigned to the variable x.
- Following are invalid expressions:
  4=x;
  3=4;
  a+3=5;
- Left operand must be a variable.
- Compound Assignment Operators(+=, -=, *=, /=, %=, &=, |=, ^=)

Find the output of the following program:



```
int main()
{
    int x=5;
    x+=4; //same as x=x+4
    printf("x=%d",x);
    return(0);
}
```

Output is

x=9

Expression x+=4 means, x is incremented by 4.

Similarly,

x-=3; is same as x=x-3;

x*=5; is same as x=x*5;

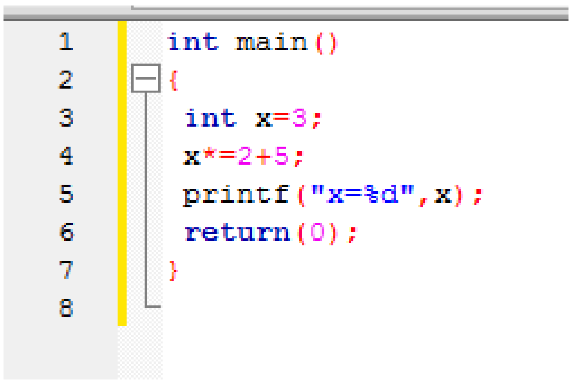
x/=7; is same as x=x/7;

x%=3; is same as x=x%3;

Caution

Do not interpret that the expression x*=2+5 can be simply replaced by x=x*2+5, because there is a difference in priority. To understand this concept observe the following two examples

Find output of the following code:



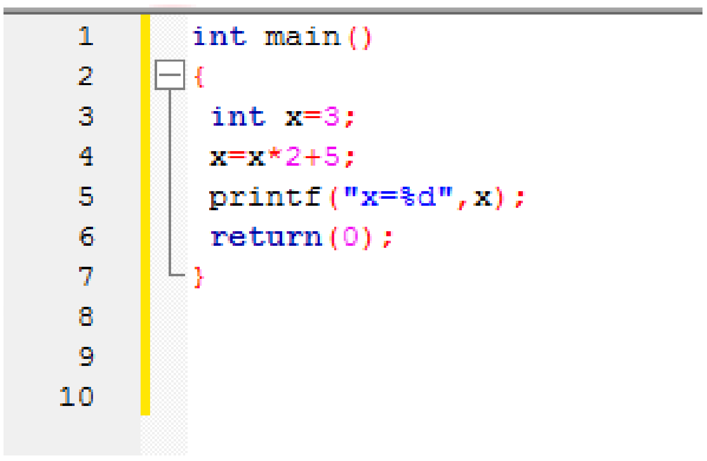
```c
int main()
{
 int x=3;
 x*=2+5;
 printf("x=%d",x);
 return(0);
}
```

Output is

x=21

In the expression x*=2+5; operator + has higher priority, thus resolved as x*=7, which further results 21.

Find output of the following code:



```c
int main()
{
 int x=3;
 x=x*2+5;
 printf("x=%d",x);
 return(0);
}
```

Output is

x=11

In the expression x=x*2+5; operator * has the highest priority, thus the expression is resolved as x=6+5, which is further evaluated as x=11.

References

YouTube video links

- Lecture 5 Arithmetic Instruction part-1
  - https://www.youtube.com/watch?v=CeEEv-qiMTc&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW

- Lecture 5 Arithmetic Instruction part-2
  - https://www.youtube.com/watch?v=5UOMX4s_xK8&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW
- Lecture 5 Arithmetic Instruction part-3
  - https://www.youtube.com/watch?v=WLCVGSfJSYk&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW
- Lecture 5 Arithmetic Instruction part-4
  - https://www.youtube.com/watch?v=6JR-hppyfug&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW
- Bitwise NOT(~) operator
  - https://www.youtube.com/watch?v=dMvsXtiwqgc&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW
- Lecture 5 Arithmetic Instruction part-5
  - https://www.youtube.com/watch?v=iAtVzg42oHY&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW
- Lecture 5 Arithmetic Instruction part-6
  - https://www.youtube.com/watch?v=beO4ufKH4Bs&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW
- Lecture 5 Arithmetic Instruction part-7
  - https://www.youtube.com/watch?v=rqSyHXTOu3I&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW
- Conditional Operator
  - https://www.youtube.com/watch?v=7iV0AaNQ6Bs&feature=youtu.be&list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW

Exercise

1. Write a program to calculate average of three numbers. (if user enters 3,5,2 then your output should be 3.33)
2. Write a program to profit percentage, user input selling price and cost price of item.
3. Write a program to print 5 greater number of user input. (if user enters 7 your output should be 12)
4. Write a program to print user input number with the last digit lost. (if user enters 237, your output should be 23. If user inputs 4, your output should be 0)
5. Write a program to swap two integer variable data.
6. Write a program to swap two integer variable data without using third variable
7. Write a program to swap two integer variable data without using third variable and arithmetic operators
8. Write a program to input two digit number and your output should be reverse of number. (if user enters 45, your output should be 54)
9. Write a program to add all digits of a three digit number. Number is given by user. (if user enters 384, your output should be 15)

10. Write a program to print only last digit of a given number. (if user enters 546, your output should be 6)