**C – Operators and Expressions**

* The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
* These C operators join individual constants and variables to form.
* Operators, functions, constants and variables are combined together to form expressions.
* Consider the expression A + B \* 5. where, +, \* are operators, A, B  are variables, 5 is constant and A + B \* 5 is an expression.

**TYPES OF C OPERATORS:**

**C language offers many types of operators. They are,**

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

**CONTINUE ON TYPES OF C OPERATORS:**

Click on each operator name below for detailed description and example programs.

|  |  |
| --- | --- |
| **Types of Operators** | **Description** |
| [Arithmetic\_operators](http://fresh2refresh.com/c/c-operators-expressions/c-arithmetic-operators/) | These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus |
| [Assignment\_operators](http://fresh2refresh.com/c/c-operators-expressions/c-assignment-operators/) | These are used to assign the values for the variables in C programs. |
| [Relational operators](http://fresh2refresh.com/c/c-operators-expressions/c-relational-operators/) | These operators are used to compare the value of two variables. |
| [Logical operators](http://fresh2refresh.com/c/c-operators-expressions/c-logical-operators/) | These operators are used to perform logical operations on the given two variables. |
| [Bit wise operators](http://fresh2refresh.com/c/c-operators-expressions/c-bit-wise-operators/) | These operators are used to perform bit operations on given two variables. |
| [Conditional (ternary) operators](http://fresh2refresh.com/c/c-operators-expressions/c-conditional-operators/) | Conditional operators return one value if condition is true and returns another value is condition is false. |
| [Increment/decrement operators](http://fresh2refresh.com/c/c-operators-expressions/c-increment-decrement-operators/) | These operators are used to either increase or decrease the value of the variable by one. |
| [Special operators](http://fresh2refresh.com/c/c-operators-expressions/c-special-operators/) | &, \*, sizeof( ) and ternary operators. |

# – Arithmetic Operators

#### [PREV](http://fresh2refresh.com/c/c-operators-expressions/)     [NEXT](http://fresh2refresh.com/c/c-operators-expressions/c-assignment-operators/)

# ARITHMETIC OPERATORS IN C:

C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs.

|  |  |
| --- | --- |
| **Arithmetic Operators/Operation** | **Example** |
| + (Addition) | A+B |
| – (Subtraction) | A-B |
| \* (multiplication) | A\*B |
| / (Division) | A/B |
| % (Modulus) | A%B |

# ****EXAMPLE PROGRAM FOR C ARITHMETIC OPERATORS:****

In this example program, two values “40” and “20” are used to perform arithmetic operations such as addition, subtraction, multiplication, division, modulus and output is displayed for each operation.

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <stdio.h>    int main()  {     int a=40,b=20, add,sub,mul,div,mod;     add = a+b;     sub = a-b;     mul = a\*b;     div = a/b;     mod = a%b;     printf("Addition of a, b is : %d\n", add);     printf("Subtraction of a, b is : %d\n", sub);     printf("Multiplication of a, b is : %d\n", mul);     printf("Division of a, b is : %d\n", div);     printf("Modulus of a, b is : %d\n", mod);  } |

# ****OUTPUT:****

|  |
| --- |
| Addition of a, b is : 60 Subtraction of a, b is : 20 Multiplication of a, b is : 800 Division of a, b is : 2 Modulus of a, b is : 0 |

**ASSIGNMENT OPERATORS IN C:**

In C programs, values for the variables are assigned using assignment operators.

* For example, if the value “10” is to be assigned for the variable “sum”, it can be assigned as “sum = 10;”
* There are 2 categories of assignment operators in C language. They are,  
  1. Simple assignment operator ( Example: = )  
  2. Compound assignment operators ( Example: +=, -=, \*=, /=, %=, &=, ^= )

|  |  |
| --- | --- |
| **Operators** | **Example/Description** |
| = | sum = 10; 10 is assigned to variable sum |
| += | sum += 10;  This is same as sum = sum + 10 |
| -= | sum -= 10;  This is same as sum = sum – 10 |
| \*= | sum \*= 10;  This is same as sum = sum \* 10 |
| /= | sum /= 10;  This is same as sum = sum / 10 |
| %= | sum %= 10;  This is same as sum = sum % 10 |
| &= | sum&=10;  This is same as sum = sum & 10 |
| ^= | sum ^= 10;  This is same as sum = sum ^ 10 |

**C – Relational Operators**

[**PREV**](http://fresh2refresh.com/c/c-operators-expressions/c-assignment-operators/)[**NEXT**](http://fresh2refresh.com/c/c-operators-expressions/c-logical-operators/)

**RELATIONAL OPERATORS IN C:**

Relational operators are used to find the relation between two variables. i.e. to compare the values of two variables in a C program.

|  |  |
| --- | --- |
| **Operators** | **Example/Description** |
| > | x > y (x is greater than y) |
| < | x < y (x is less than y) |
| >= | x >= y (x is greater than or equal to y) |
| <= | x <= y (x is less than or equal to y) |
| == | x == y (x is equal to y) |
| != | x != y (x is not equal to y) |

**EXAMPLE PROGRAM FOR RELATIONAL OPERATORS IN C:**

* In this program, relational operator (==) is used to compare 2 values whether they are equal are not.
* If both values are equal, output is displayed as ” values are equal”. Else, output is displayed as “values are not equal”.
* Note : double equal sign (==) should be used to compare 2 values. We should not single equal sign (=).

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | #include <stdio.h>    int main()  {     int m=40,n=20;     if (m == n)     {         printf("m and n are equal");     }     else     {         printf("m and n are not equal");     }  } |

**OUTPUT:**

|  |
| --- |
| m and n are not equal |

**LOGICAL OPERATORS IN C:**

These operators are used to perform logical operations on the given expressions.

* There are 3 logical operators in C language. They are, logical AND (&&), logical OR (||) and logical NOT (!).

|  |  |
| --- | --- |
| **Operators** | **Example/Description** |
| && (logical AND) | (x>5)&&(y<5) It returns true when both conditions are true |
| || (logical OR) | (x>=10)||(y>=10)  It returns true when at-least one of the condition is true |
| ! (logical NOT) | !((x>5)&&(y<5))  It reverses the state of the operand “((x>5) && (y<5))”  If “((x>5) && (y<5))” is true, logical NOT operator makes it false |

**EXAMPLE PROGRAM FOR LOGICAL OPERATORS IN C:**

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include <stdio.h>    int main()  {     int m=40,n=20;     int o=20,p=30;     if (m>n && m !=0)     {        printf("&& Operator : Both conditions are true\n");     }     if (o>p || p!=20)     {        printf("|| Operator : Only one condition is true\n");     }     if (!(m>n && m !=0))     {        printf("! Operator : Both conditions are true\n");     }     else     {        printf("! Operator : Both conditions are true. " \        "But, status is inverted as false\n");     }  } |

**OUTPUT:**

|  |
| --- |
| && Operator : Both conditions are true || Operator : Only one condition is true ! Operator : Both conditions are true. But, status is inverted as false |

**C – Bit wise Operators**

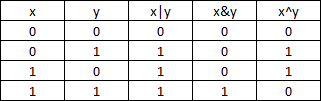
[**PREV**](http://fresh2refresh.com/c/c-operators-expressions/c-logical-operators/)[**NEXT**](http://fresh2refresh.com/c/c-operators-expressions/c-conditional-operators/)

**BIT WISE OPERATORS IN C:**

These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.

* Bit wise operators in C language are & (bitwise AND), | (bitwise OR), ~ (bitwise NOT), ^ (XOR), << (left shift) and >> (right shift).

**TRUTH TABLE FOR BIT WISE OPERATION & BIT WISE OPERATORS:**



**BELOW ARE THE BIT-WISE OPERATORS AND THEIR NAME IN C LANGUAGE.**

1. & – Bitwise AND
2. | – Bitwise OR
3. ~ – Bitwise NOT
4. ^ – XOR
5. << – Left Shift
6. >> – Right Shift

Consider x=40 and y=80. Binary form of these values are given below.

x = 00101000  
y=  01010000

All bit wise operations for x and y are given below.

1. x&y = 00000000 (binary) = 0 (decimal)
2. x|y = 01111000 (binary) = 120 (decimal)
3. ~x = 11111111111111111111111111 11111111111111111111111111111111010111 = -41 (decimal)
4. x^y = 01111000 (binary) = 120 (decimal)
5. x << 1 = 01010000 (binary) = 80 (decimal)
6. x >> 1 = 00010100 (binary) = 20 (decimal)

**NOTE:**

* **Bit wise NOT :** Value of 40 in binary is 00000000000000000000000000000000 00000000000000000010100000000000. So, all 0’s are converted into 1’s in bit wise NOT operation.
* **Bit wise left shift and right shift :** In left shift operation “x << 1 “, 1 means that the bits will be left shifted by one place. If we use it as “x << 2 “,  then, it means that the bits will be left shifted by 2 places.

**EXAMPLE PROGRAM FOR BIT WISE OPERATORS IN C:**

In this example program, bit wise operations are performed as shown above and output is displayed in decimal format.

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <stdio.h>    int main()  {     int m = 40,n = 80,AND\_opr,OR\_opr,XOR\_opr,NOT\_opr ;     AND\_opr = (m&n);     OR\_opr = (m|n);     NOT\_opr = (~m);     XOR\_opr = (m^n);     printf("AND\_opr value = %d\n",AND\_opr );     printf("OR\_opr value = %d\n",OR\_opr );     printf("NOT\_opr value = %d\n",NOT\_opr );     printf("XOR\_opr value = %d\n",XOR\_opr );     printf("left\_shift value = %d\n", m << 1);     printf("right\_shift value = %d\n", m >> 1);  } |

**OUTPUT:**

|  |
| --- |
| AND\_opr value = 0 OR\_opr value = 120 NOT\_opr value = -41 XOR\_opr value = 120 left\_shift value = 80 right\_shift value = 20 |

**C – Conditional Operators**

[**PREV**](http://fresh2refresh.com/c/c-operators-expressions/c-bit-wise-operators/)[**NEXT**](http://fresh2refresh.com/c/c-operators-expressions/c-increment-decrement-operators/)

**CONDITIONAL OPERATORS IN C:**

* Conditional operators return one value if condition is true and returns another value is condition is false.
* This operator is also called as ternary operator.

Syntax     :        (Condition? true\_value: false\_value);

Example :         (A > 100  ?  0  :  1);

* In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

**EXAMPLE PROGRAM FOR CONDITIONAL/TERNARY OPERATORS IN C:**

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include <stdio.h>    int main()  {     int x=1, y ;     y = ( (x ==1) && (y != 3)) ? a+3+2/3\*2 : 0 ) ;     printf("x value is %d\n", x);     printf("y value is %d", y);  } |

**OUTPUT:**

|  |
| --- |
| x value is 1 y value is 2 |

**C – Increment/decrement Operators**

[**PREV**](http://fresh2refresh.com/c/c-operators-expressions/c-conditional-operators/)[**NEXT**](http://fresh2refresh.com/c/c-operators-expressions/c-special-operators/)

Increment/decrement Operators in C:

* Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.
* Syntax:  
  Increment operator: ++var\_name; (or) var\_name++;  
  Decrement operator: – -var\_name; (or) var\_name – -;
* Example:  
  Increment operator :  ++ i ;    i ++ ;  
  Decrement operator :  – – i ;   i – – ;

**EXAMPLE PROGRAM FOR INCREMENT OPERATORS IN C:**

In this program, value of “i” is incremented one by one from 1 up to 9 using “i++” operator and output is displayed as “1 2 3 4 5 6 7 8 9”.

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | //Example for increment operators    #include <stdio.h>  int main()  {       int i=1;       while(i<10)       {           printf("%d ",i);           i++;       }  } |

**OUTPUT:**

|  |
| --- |
| 1 2 3 4 5 6 7 8 9 |

# C – Special Operators

#### [PREV](http://fresh2refresh.com/c/c-operators-expressions/c-increment-decrement-operators/)     [NEXT](http://fresh2refresh.com/c/c-decision-control/)

# ****SPECIAL OPERATORS IN C:****

Below are some of the special operators that the C programming language offers.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| & | This is used to get the address of the variable.  Example : &a will give address of a. |
| \* | This is used as pointer to a variable.  Example : \* a  where, \* is pointer to the variable a. |
| Sizeof () | This gives the size of the variable.  Example : size of (char) will give us 1. |

# ****EXAMPLE PROGRAM FOR & AND \* OPERATORS IN C:****

In this program, “&” symbol is used to get the address of the variable and “\*” symbol is used to get the value of the variable that the pointer is pointing to. Please refer [**C – pointer**](http://fresh2refresh.com/c/c-pointer/) topic to know more about pointers.

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include <stdio.h>  int main()  {  int \*ptr, q;  q = 50;  /\* address of q is assigned to ptr \*/  ptr = &q;  /\* display q's value using ptr variable \*/  printf("%d", \*ptr);  return 0;  } |

# ****OUTPUT:****

|  |
| --- |
| 50 |

# ****EXAMPLE PROGRAM FOR SIZEOF() OPERATOR IN C:****

sizeof() operator is used to find the memory space allocated for each C data types.

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #include <stdio.h>  #include <limits.h>    int main()  {  int a;  char b;  float c;  double d;  printf("Storage size for int data type:%d \n",sizeof(a));  printf("Storage size for char data type:%d \n",sizeof(b));  printf("Storage size for float data type:%d \n",sizeof(c));  printf("Storage size for double data type:%d\n",sizeof(d));  return 0;  } |

# ****OUTPUT:****

|  |
| --- |
| Storage size for int data type:4  Storage size for char data type:1 Storage size for float data type:4 Storage size for double data type:8 |

**C – Tokens and keywords**

C tokens, Identifiers and Keywords are the basics in a C program.

**1. C TOKENS**

* C tokensare the basic buildings blocks in C language which are constructed together to write a C program.
* Each and every smallest individual units in a C program are known as C tokens.

**C tokens are of six types. They are,**

1. Keywords               (eg: int, while),
2. Identifiers               (eg: main, total),
3. Constants              (eg: 10, 20),
4. Strings                    (eg: “total”, “hello”),
5. Special symbols   (eg: (), {}),
6. Operators               (eg: +, /,-,\*)

**C TOKENS EXAMPLE PROGRAM:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int main()  {     int x, y, total;     x = 10, y = 20;     total = x + y;     printf ("Total = %d \n", total);  } |

where,

* main – identifier
* {,}, (,) – delimiter
* int – keyword
* x, y, total – identifier
* main, {, }, (, ), int, x, y, total – tokens

Do you know how to use C token in real time application programs? We have given simple real time application programs where C token is used. You can refer the below C programs to know how to use C token in real time program.

**2. IDENTIFIERS IN C LANGUAGE:**

* Each program elements in a C program are given a name called identifiers.
* Names given to identify Variables, functions and arrays are examples for identifiers. eg. x is a name given to integer variable in above program.

**RULES FOR CONSTRUCTING IDENTIFIER NAME IN C:**

1. First character should be an alphabet or underscore.
2. Succeeding characters might be digits or letter.
3. Punctuation and special characters aren’t allowed except underscore.
4. Identifiers should not be keywords.

**3. KEYWORDS IN C LANGUAGE:**

* Keywords are pre-defined words in a C compiler.
* Each keyword is meant to perform a specific function in a C program.
* Since keywords are referred names for compiler, they can’t be used as variable name.

C language supports 32 keywords which are given below. Click on each keywords below for detail description and example programs.

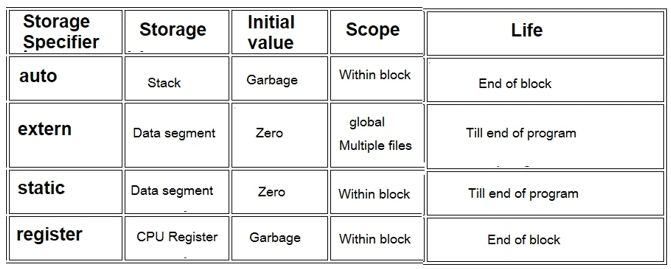
|  |  |
| --- | --- |
| [auto](http://fresh2refresh.com/c/c-storage-class-specifiers/) | [double](http://fresh2refresh.com/c/c-data-types/) |
| [int](http://fresh2refresh.com/c/c-data-types/) | [struct](http://fresh2refresh.com/c/c-structures/) |
| [const](http://fresh2refresh.com/c/c-constants/) | [float](http://fresh2refresh.com/c/c-data-types/) |
| [short](http://fresh2refresh.com/c/c-data-types/) | [unsigned](http://fresh2refresh.com/c/c-data-types/) |
| [break](http://fresh2refresh.com/c/c-case-control-statements/) | [else](http://fresh2refresh.com/c/c-decision-control/) |
| [long](http://fresh2refresh.com/c/c-data-types/) | [switch](http://fresh2refresh.com/c/c-case-control-statements/) |
| [continue](http://fresh2refresh.com/c/c-case-control-statements/) | [for](http://fresh2refresh.com/c/c-loop-control-statements/) |
| [signed](http://fresh2refresh.com/c/c-data-types/) | [void](http://fresh2refresh.com/c/c-data-types/) |

|  |  |
| --- | --- |
| [case](http://fresh2refresh.com/c/c-case-control-statements/) | [enum](http://fresh2refresh.com/c/c-data-types/) |
| [register](http://fresh2refresh.com/c/c-storage-class-specifiers/) | [typedef](http://fresh2refresh.com/c/c-typedef/) |
| [default](http://fresh2refresh.com/c/c-case-control-statements/) | [goto](http://fresh2refresh.com/c/c-case-control-statements/) |
| [sizeof](http://fresh2refresh.com/c/c-data-types/) | [volatile](http://fresh2refresh.com/c/c-type-qualifiers/) |
| [char](http://fresh2refresh.com/c/c-data-types/) | [extern](http://fresh2refresh.com/c/c-storage-class-specifiers/) |
| [return](http://fresh2refresh.com/c/c-function/c-function-arguments-and-return-values/) | [union](http://fresh2refresh.com/c/c-union/) |
| [do](http://fresh2refresh.com/c/c-loop-control-statements/) | [if](http://fresh2refresh.com/c/c-decision-control/) |
| [static](http://fresh2refresh.com/c/c-storage-class-specifiers/) | [while](http://fresh2refresh.com/c/c-loop-control-statements/) |

## Introduction to C Storage Classes

Every value or number needs to be stored in someplace for later usage, right? This can be done using variables in C. Variables are storage areas used in our programs. Each variable will be of a specific type like integer, character and also will have specific size and layout depending on their type. Each type of variable should be stored in specific part of the memory and will have restricted or specific access permissions.

Storage Classes in C determines in which part of the memory each variable should be stored and also it decides the scope (visibility) of the variable. There are four types of storage classes–auto, extern, static and register. One storage specifier can be mentioned along with the variable.



Types of Storage Classes in C are as follows

#### 1. Automatic Storage Class

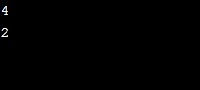
All variables declared within a function or block, will be stored in an auto specifier by default even if it is not explicitly defined. The specifier for this storage class is ‘auto’. The scope or visibility of the variables in automatic storage class is local to the block or function it is defined. The variable will be destroyed once we come out of the function or the block.

This can be explained better with an example. Consider the example given below:

**Code:**

#include<stdio.h>  
int main() {  
int i = 2;  
{ int i = 4;  
printf("%d\n", i);  
}  
printf("%d\n", i);  
}

**Output:**



Here, a variable I of type integer is declared first with value 2 assigned to it. Next inside a loop or block again variable I of the same integer type is declared with value 4 assigned to it. If storage specifier is not mentioned, by default it will be taken as auto. In the first printf statement, which is mentioned inside the block will print 4 on printing the value of I. Whereas in the second printf statement which is mentioned outside the block, will print the value of I as 2, the value which is mentioned outside the block. It is better to initial some value to auto variables because there are chances of getting any garbage value sometimes if initialization is not done. This example gives a clear picture of auto variables and about local scope.

#### 2. Register Storage Class

The variables stored in the register storage class will also have local scope, which means it is accessible or visible only in the block in which it is declared. This storage is similar to auto, but the main difference is that auto variables are stored in memory whereas the register variables are stored in the CPU registers. This is done if we want to access the variable very frequently. These can be used faster. Only a few variables are stored using register specifiers. If there is no space in register then it is stored in the memory only. No initial value is assigned to the register variables. Also, & (address-of) operator cannot be used on register variables. For example, variables which are used for counters or similar usage types are stored using register specifier.

#### 3. Static Storage Class

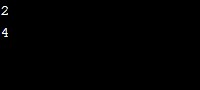
Variable, may it be global or local, are stored using static specifier in static storage class when the variable needs to be declared once and the value needs to be retained. When a variable is declared as static, the value will be saved or retained between the function calls.  Permanent storage is created and it is declared only once. When a  local variable is declared as static, permanent storage is created for it and the value is retained every time it is used. Also as per the scope of the usual local variable, static local variables are also visible only to the function or block where it is defined. When a global variable is declared as static, similar to static local, permanent storage is created and it is declared only once. But even though it is global, these variables are only visible within the file in which it is defined.

Static variables can be clearly pictured using the below example:

**Code:**

#include<stdio.h>  
int samplefunc() {  
static int a = 0;  
a = a+2;  
return a;  
}  
int main() {  
int result1 = samplefunc();  
int result2 = samplefunc();  
printf("%d\n", result1);  
printf("%d\n", result2);  
}

**Output:**



Here, in the above program, when the samplefunc() is called, the variable a is defined and initialized the first time and permanent storage is created for it. By the mathematical expression used in the function the value of a then becomes 2. But when the same samplefunc() is called the second time, variable a is not defined or initialized again, rather it takes the last retained value and continues with the operation making the final result as 4. This is the main usage and advantage of static variables.

#### 4. Extern Storage Class

Variable declared as extern, depicts that the variable is defined elsewhere in another program. These extern variables are used when we want any variable or function defined in one program to be used in another file too. The variables with extern specifier are stored in the extern storage class. When the variable is declared as extern is a program, it specifies the external linkage, and hence it is not defined or initialized again. Storage is allocated only one and also initialized only once. If extern variables are initialized again with another value in the external program, we will get an error stating ‘Redefinition of the variable’.

Extern variables are explained using the below example:

**Code:**

**Prg1.c**

int count;  
int main() {

int count.  
count = 10;  
}

**Prg2.c**

extern int count;  
int main() {  
printf(“%d”, count);  
}

**Output:**

C Storage Classes-3

Here, the integer variable count is declared in the first C program (Prg1.c) and inside the main function, it is initialized to value 10. In the second C program, the same count variable is declared using extern specifier, which specifies that there is external linkage and the value is fetched from the storage and the value 10 is given to the value count when we print it in the second program. This is the usage of extern variables. Thus, depending on the different purpose each [storage classes are used](https://www.educba.com/storage-class-in-c-plus-plus/) for appropriate variables and it is declared with the corresponding specifiers.