

CS 240 Programming in C

Operators, Bitwise, Header Files

September 22, 2022

Operators in C

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators ¹

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

¹Link: https://www.tutorialspoint.com/cprogramming/c_operators.htm

Arithmetic Operators

Assume variable A holds 10 and variable B holds 20 then

Operator	Description	Example
+	Adds two operands.	$A + B = 30$
-	Subtracts second operand from the first.	$A - B = -10$
*	Multiplies both operands.	$A * B = 200$
/	Divides numerator by de-numerator.	$B / A = 2$
%	Modulus Operator and remainder of after an integer division.	$B \% A = 0$
++	Increment operator increases the integer value by one.	$A++ = 11$
--	Decrement operator decreases the integer value by one.	$A-- = 9$

Relational Operators

Assume variable A holds 10 and variable B holds 20 then

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Assume variable A holds 1 and variable B holds 0, then

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation.

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation.

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	$(A \& B) = 12$, i.e., 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	$(A B) = 61$, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	$(A \wedge B) = 49$, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	$(\sim A) = \sim(60)$, i.e., -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	$A \ll 2 = 240$ i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	$A \gg 2 = 15$ i.e., 0000 1111

Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	$C = A + B$ will assign the value of $A + B$ to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	$C /= A$ is equivalent to $C = C / A$
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	$C \% = A$ is equivalent to $C = C \% A$
<<=	Left shift AND assignment operator.	$C <<= 2$ is same as $C = C << 2$
>>=	Right shift AND assignment operator.	$C >>= 2$ is same as $C = C >> 2$
&=	Bitwise AND assignment operator.	$C \&= 2$ is same as $C = C \& 2$
^=	Bitwise exclusive OR and assignment operator.	$C \wedge= 2$ is same as $C = C \wedge 2$
=	Bitwise inclusive OR and assignment operator.	$C = 2$ is same as $C = C 2$

Misc Operators

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

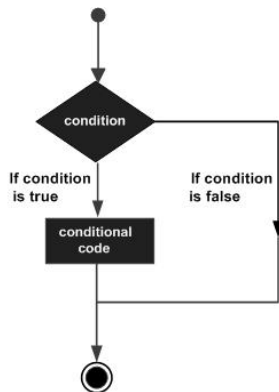
Operators Precedence Table

Reference on Chapter 2.12.

Decision Making

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Decision Making



C programming language assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value.

The ? : Operator

`Exp1 ? Exp2 : Exp3;`

Where `Exp1`, `Exp2`, and `Exp3` are expressions. Notice the use and placement of the colon.

- The value of a `?` expression is determined like this
- `Exp1` is evaluated. If it is true, then `Exp2` is evaluated and becomes the value of the entire `?` expression.
- If `Exp1` is false, then `Exp3` is evaluated and its value becomes the value of the expression.

Library Functions in Different Header Files

- `assert.h` - Program assertion functions
- `ctype.h` - Character type functions
- `locale.h` - Localization functions
- `math.h` - Mathematics functions
- `setjmp.h` - Jump functions
- `signal.h` - Signal handling functions
- `stdarg.h` - Variable arguments handling functions
- `stdio.h` - Standard Input/Output functions
- `stdlib.h` - Standard Utility functions
- `string.h` - String handling functions
- `time.h` - Date time functions

Ctype.h Functions

Function	Return Type	Use
isalnum(c)	int	Determine if the argument is alphanumeric or not
isalpha(c)	int	Determine if the argument is alphabetic or not
isascii(c)	int	Determine if the argument is ASCII character or not
isdigit(c)	int	Determine if the argument is a decimal digit or not.
toascii(c)	int	Convert value of argument to ASCII
tolower(c)	int	Convert character to lower case
toupper(c)	int	Convert letter to uppercase

2

²<https://www.startertutorials.com/blog/functions-in-c.html>

Math.h Functions

Function	Return Type	Use
<code>ceil(d)</code>	double	Returns a value rounded up to next higher integer
<code>floor(d)</code>	double	Returns a value rounded up to next lower integer
<code>cos(d)</code>	double	Returns the cosine of d
<code>sin(d)</code>	double	Returns the sine of d
<code>tan(d)</code>	double	Returns the tangent of d
<code>exp(d)</code>	double	Raise e to the power of d
<code>fabs(d)</code>	double	Returns the absolute value of d
<code>pow(d1, d2)</code>	double	Returns d1 raised to the power of d2
<code>sqrt(d)</code>	double	Returns the square root of d

Stdlib.h Functions

Function	Return Type	Use
<code>abs(i)</code>	<code>int</code>	Return the absolute value of <code>i</code>
<code>exit(u)</code>	<code>void</code>	Close all file and buffers, and terminate the program
<code>rand(void)</code>	<code>int</code>	Return a random positive integer
<code>calloc(u1, u2)</code>	<code>void*</code>	Allocate memory for an array having <code>u1</code> elements, each of length <code>u2</code> bytes
<code>malloc(u)</code>	<code>void*</code>	Allocate <code>u</code> bytes of memory
<code>realloc(p,u)</code>	<code>void*</code>	Allocate <code>u</code> bytes of new memory to the pointer variable <code>p</code>
<code>free(p)</code>	<code>void</code>	Free a block of memory whose beginning is indicated by <code>p</code>

String.h Functions

Function	Return Type	Use
<code>strcmp(s1,s2)</code>	<code>int</code>	Compare two strings
<code>strcpy(s1,s2)</code>	<code>char*</code>	Copy string s2 to s1
<code>strlen(s)</code>	<code>int</code>	Return the number of characters in string s
<code>strrev(s)</code>	<code>char*</code>	Return the reverse of the string s

time.h Functions

Function	Return type	Use
<code>difftime(11,12)</code>	double	Return the difference between 11 ~ 12.
<code>time(p)</code>	long int	Return the number of seconds elapsed beyond a designated base time