Year	Subject Title	Sem	Sub Code
2018 -19 Onwards	Core: COMPUTER FUNDAMENTALS AND C PROGRAMMING	I	18BIT13C

UNIT II: Overview of C: Introduction - Character set - C tokens - keyword & Identifiers -Constants - Variables - Data types - Declaration of variables - Assigning values to variables -Defining Symbolic Constants - Reading & Writing a character - Formatted input and output - Arithmetic, Relational, Logical, Assignment, Increment and Decrement operators, Conditional, Bitwise, Special Operators - Arithmetic Expressions - Evaluation of expressions -precedence of arithmetic operators - Type conversion in expressions – operator precedence &associatively - Mathematical functions.

TEXT BOOK

E. Balagurusamy," Computing Fundamentals & C Programming" – Tata McGraw-Hill, Second Reprint 2008, ISBN 978-0-07-066909-3.

Prepared by P.Sundari.

C - CHARACTER SET

As every language contains a set of characters used to construct words, statements, etc., C language also has a set of characters which include **alphabets, digits**, and **special symbols**. C language supports a total of 256 characters.

Every C program contains statements. These statements are constructed using words and these words are constructed using characters from C character set. C language character set contains the following set of characters...

1. Alphabets
2. Digits
3. Special Symbols

Alphabets

C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.

lower case letters - **a to z**

UPPER CASE LETTERS - **A to Z**

Digits

C language supports 10 digits which are used to construct numerical values in C language.

Digits - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

Special Symbols

C language supports a rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, backspaces, and other special symbols.

Special Symbols - **~ @ # $ % ^ & * ( ) _ - + = { } [ ] ; : ' " / ? . > , < \ | tab newline space NULL bell backspace vertical tab etc.,**

Tokens in C

Tokens are the smallest elements of a program, which are meaningful to the compiler.

The following are the types of tokens: Keywords, Identifiers, Constant, Strings, Operators, etc.

Let us begin with Keywords.

Keywords

Keywords are predefined, reserved words in C and each of which is associated with specific features. These words help us to use the functionality of C language. They have special meaning to the compilers.

There are total 32 keywords in C.

Auto	double	int	struct
Break	else	long	switch
Case	enum	register	typedef
Char	extern	return	union

Continue	for	signed	void
Do	if	static	while
Default	goto	sizeof	volatile
Const	float	short	unsigned

Identifiers

Each program element in C programming is known as an identifier. They are used for naming of variables, functions, array etc. These are user-defined names which consist of alphabets, number, underscore '_'. Identifier's name should not be same or same as keywords. Keywords are not used as identifiers.

Rules for naming C identifiers –

- It must begin with alphabets or underscore.
- Only alphabets, numbers, underscore can be used, no other special characters, punctuations are allowed.
- It must not contain white-space.
- It should not be a keyword.
- It should be up to 31 characters long.

Strings

A string is an array of characters ended with a null character(\0). This null character indicates that string has ended. Strings are always enclosed with double quotes(" ").

Let us see how to declare String in C language –

- char string[20] = {'s','t','u','d','y', '\0'};
- char string[20] = "demo";
- char string [] = "demo";

C - Constants and Literals

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are enumeration constants as well.

Constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals −

```
212         /* Legal */
215u        /* Legal */
0xFeeL      /* Legal */
078         /* Illegal: 8 is not an octal digit */
032UU       /* Illegal: cannot repeat a suffix */
```

Following are other examples of various types of integer literals −

```
85         /* decimal */
0213       /* octal */
0x4b       /* hexadecimal */
30         /* int */
30u        /* unsigned int */
30l        /* long */
30ul       /* unsigned long */
```

Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing decimal form, you must include the decimal point, the exponent, or both; and while representing exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Here are some examples of floating-point literals −

```
3.14159       /* Legal */
314159E-5L    /* Legal */
510E          /* Illegal: incomplete exponent */
210f          /* Illegal: no decimal or exponent */
.e55          /* Illegal: missing integer or fraction */
```

Character Constants

Character literals are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C that represent special meaning when preceded by a backslash for example, newline (\n) or tab (\t).

String Literals

String literals or constants are enclosed in double quotes "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating them using white spaces.

C - Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase

Sr.No.	Type & Description
1	**char** Typically a single octet(one byte). It is an integer type.
2	**int** The most natural size of integer for the machine.
3	**float** A single-precision floating point value.
4	**double** A double-precision floating point value.
5	**void** Represents the absence of type.

letters are distinct because C is case-sensitive. Based on the basic types explained in the previous chapter, there will be the following basic variable types –

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

Variable Definition in C

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows −

```
type variable_list;
```

Here, **type** must be a valid C data type including char, w_char, int, float, double, bool, or any user-defined object; and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here −

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** declares and defines the variables i, j, and k; which instruct the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows −

```
type variable_name = value;
```

Some examples are −

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables are undefined.

Variable Declaration in C

A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation without requiring the complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use the keyword **extern** to declare a variable at any place. Though you can declare a variable multiple times in your C program, it can be defined only once in a file, a function, or a block of code.

C - Data Types

Data types in C refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows −

Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void** The type specifier *void* indicates that no value is available.
4	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges –

Type	**Storage size**	**Value range**
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions *sizeof(type)* yields the storage size of the object or type in bytes.

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision −

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs.

Assigning values to variables

There are two kinds of expressions in C −

- **lvalue** − Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue** − The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.

Variables are lvalues and so they may appear on the left-hand side of an assignment. Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side. Take a look at the following valid and invalid statements −

```
int g = 20; // valid statement

10 = 20; // invalid statement; would generate compile-time error
```

Symbolic Constant

Symbolic constant is name that substitute for a sequence of character that cannot be changed. The character may represent a numeric constant, a character constant, or a string. When the program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence. They are usually defined at the beginning of the program. The symbolic constants may then appear later in the program in place of the numeric constants, character constants, etc., that the symbolic constants represent.

For example

A **C program** consists of the following symbolic constant definitions.

```
#define    PI         3.141593
#define    TRUE        1
#define    FALSE       0
```

#define PI 3.141593 defines a symbolic constant PI whose value is 3.141593. When the program is preprocessed, all occurrences of the symbolic constant PI are replaced with the replacement text 3.141593.

C - Operators

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 −

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

We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language.

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

The following table shows all the relational operators supported by C. 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

Following table shows all the logical operators supported by C language. 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. The truth tables for &, |, and ^ is as follows −

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

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

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 ^ B) = 49, i.e., 0011 0001

~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(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 << 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 >> 2 = 15 i.e., 0000 1111

Assignment Operators

The following table lists the assignment operators supported by the C language −

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 ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

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 in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associativity**
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left

Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

C Expressions

An expression is a formula in which operands are linked to each other by the use of operators to compute a value. An operand can be a function reference, a variable, an array element or a constant.

Let's see an example:

```
a-b;
```

In the above expression, minus character (-) is an operator, and a, and b are the two operands.

There are four types of expressions exist in C:

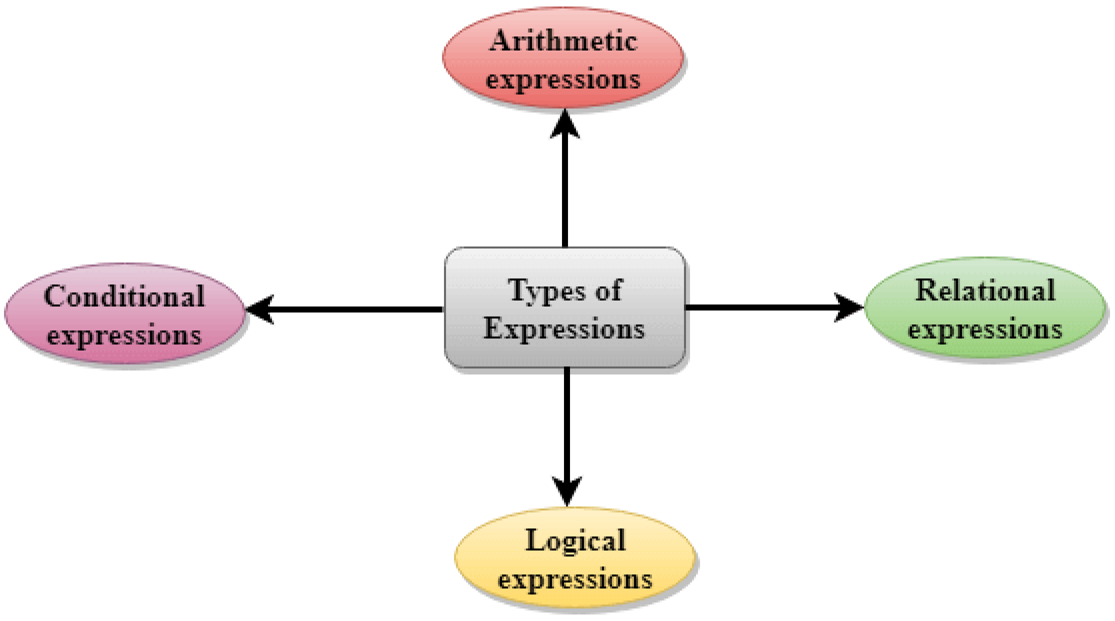
- Arithmetic expressions
- Relational expressions
- Logical expressions
- Conditional expressions

Each type of expression takes certain types of operands and uses a specific set of operators. Evaluation of a particular expression produces a specific value.

For example:

1. x = 9/2 + a-b;

The entire above line is a statement, not an expression. The portion after the equal is an expression.



Arithmetic Expressions

An arithmetic expression is an expression that consists of operands and arithmetic operators. An arithmetic expression computes a value of type int, float or double.

When an expression contains only integral operands, then it is known as pure integer expression when it contains only real operands, it is known as pure real expression, and when it contains both integral and real operands, it is known as mixed mode expression.

Evaluation of Arithmetic Expressions

The expressions are evaluated by performing one operation at a time. The precedence and associativity of operators decide the order of the evaluation of individual operations.

When individual operations are performed, the following cases can be happened:

- When both the operands are of type integer, then arithmetic will be performed, and the result of the operation would be an integer value. For example, 3/2 will yield 1 not 1.5 as the fractional part is ignored.
- When both the operands are of type float, then arithmetic will be performed, and the result of the operation would be a real value. For example, 2.0/2.0 will yield 1.0, not 1.

- If one operand is of type integer and another operand is of type real, then the mixed arithmetic will be performed. In this case, the first operand is converted into a real operand, and then arithmetic is performed to produce the real value. For example, 6/2.0 will yield 3.0 as the first value of 6 is converted into 6.0 and then arithmetic is performed to produce 3.0.

Let's understand through an example.

6*2/ (2+1 * 2/3 + 6) + 8 * (8/4)

Evaluation of expression	Description of each operation
6*2/(2+1 * 2/3 +6) +8 * (8/4)	An expression is given.
6*2/(2+2/3 + 6) + 8 * (8/4)	2 is multiplied by 1, giving value 2.
6*2/(2+0+6) + 8 * (8/4)	2 is divided by 3, giving value 0.
6*2/ 8+ 8 * (8/4)	2 is added to 6, giving value 8.
6*2/8 + 8 * 2	8 is divided by 4, giving value 2.
12/8 +8 * 2	6 is multiplied by 2, giving value 12.
1 + 8 * 2	12 is divided by 8, giving value 1.
1 + 16	8 is multiplied by 2, giving value 16.
17	1 is added to 16, giving value 17.

Relational Expressions

- A relational expression is an expression used to compare two operands.
- It is a condition which is used to decide whether the action should be taken or not.
- In relational expressions, a numeric value cannot be compared with the string value.

- The result of the relational expression can be either zero or non-zero value. Here, the zero value is equivalent to a false and non-zero value is equivalent to true.

Relational Expression	Description
x%2 = = 0	This condition is used to check whether the x is an even number or not. results in value 1 if x is an even number otherwise results in value 0.
a!=b	It is used to check whether a is not equal to b. This relational expressi equal to b otherwise 0.
a+b = = x+y	It is used to check whether the expression "a+b" is equal to the expressi
a>=9	It is used to check whether the value of a is greater than or equal to 9.

Logical Expressions

- A logical expression is an expression that computes either a zero or non-zero value.
- It is a complex test condition to take a decision.

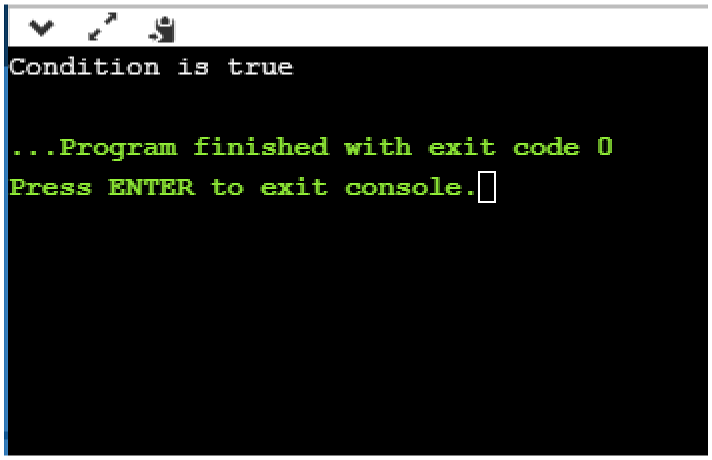
Let's see some example of the logical expressions.

Logical Expressions	Description
(x > 4) && (x < 6)	It is a test condition to check whether the x is greater than 4 and x is less than 6. the condition is true only when both the conditions are true.
x > 10 || y <11	It is a test condition used to check whether x is greater than 10 or y is less than 1 of the test condition is true if either of the conditions holds true value.
! (x > 10) && (y = = 2)	It is a test condition used to check whether x is not greater than 10 and y is eq result of the condition is true if both the conditions are true.

Let's see a simple program of "&&" operator.

```
#include <stdio.h>
int main()
{
   int x = 4;
   int y = 10;
   if ( (x <10) && (y>5))
   {
      printf("Condition is true");
   }
   else
   printf("Condition is false");
   return 0;
}
```

Output



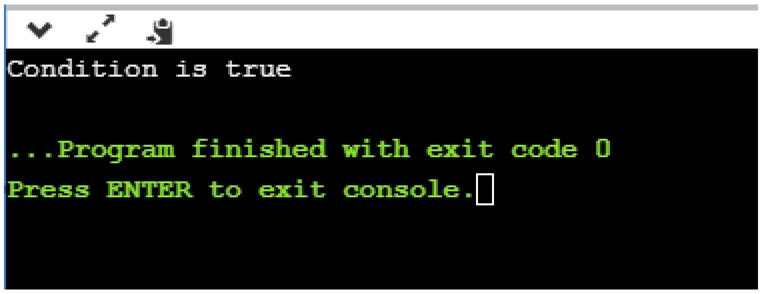
```
Condition is true

...Program finished with exit code 0
Press ENTER to exit console.
```

Let's see a simple example of "||" operator

```c
#include <stdio.h>
int main()
{
    int x = 4;
    int y = 9;
    if ( (x <6) || (y>10))
    {
        printf("Condition is true");
    }
    else
    printf("Condition is false");
    return 0;
}
```

Output



```
Condition is true

...Program finished with exit code 0
Press ENTER to exit console.
```

Conditional Expressions

- A conditional expression is an expression that returns 1 if the condition is true otherwise 0.
- A conditional operator is also known as a ternary operator.

The Syntax of Conditional operator

Suppose exp1, exp2 and exp3 are three expressions.

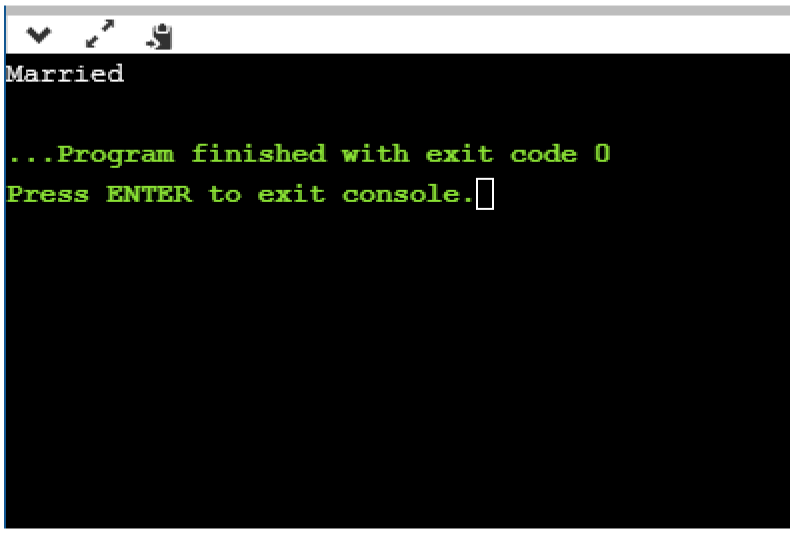
exp1 ? exp2 : exp3

The above expression is a conditional expression which is evaluated on the basis of the value of the exp1 expression. If the condition of the expression exp1 holds true, then the final conditional expression is represented by exp2 otherwise represented by exp3.

Let's understand through a simple example.

```
#include<stdio.h>
#include<string.h>
int main()
{
    int age = 25;
    char status;
    status = (age>22) ? 'M': 'U';
    if(status == 'M')
    printf("Married");
    else
    printf("Unmarried");
    return 0;
}
```

Output



```
Married

...Program finished with exit code 0
Press ENTER to exit console.
```

MANAGING I/O OPERATIONS

As we all know the three essential functions of a computer are reading, processing and writing data. Majority of the programs take data as input, and then after processing the processed data is being displayed which is called information. In C programming you can use *scanf()* and *printf()* predefined function to read and print data.

I/O operations are useful for a program to interact with users. *stdlib* is the standard C library for input-output operations. While dealing with input-output operations in C, two important streams play their role. These are:

1. Standard Input (stdin)
2. Standard Output (stdout)

Standard input or *stdin* is used for taking input from devices such as the keyboard as a data stream. *Standard output* or *stdout* is used for giving output to a device such as a monitor. For using I/O functionality, programmers must include *stdio header-file* within the program.

Reading a character in C

The easiest and simplest of all I/O operations are taking a character as input by reading that character from standard input (keyboard). *getchar()* function can be used to read a single character. This function is alternate to *scanf()* function.

Syntax:

```
var_name = getchar();
```

Example:

```
#include<stdio.h>



void main()

{

char title;
title = getchar();

}
```

There is another function to do that task for files: *getc* which is used to accept a character from standard input.
Syntax:

```
int getc(FILE *stream);
```

Writing Character In C

Similar to *getchar()* there is another function which is used to write characters, but one at a time.
Syntax:

```
putchar(var_name);
```

Example:

```
#include<stdio.h>



void main()

{

char result = 'P';

putchar(result);

putchar('\n');

}
```

Similarly, there is another function *putc* which is used for sending a single character to the standard output.
Syntax:

```
int putc(int c, FILE *stream);
```

Formatted Input

It refers to an input data which has been arranged in a specific format. This is possible in C using *scanf()*. We have already encountered this and familiar with this function.

Syntax:

scanf("control string", arg1, arg2, ..., argn);

The field specification for reading integer inputted number is:
%w sd

Here the % sign denotes the conversion specification; *w* signifies the integer number that defines the field width of the number to be read. *d* defines the number to be read in integer format.

Example:

```
#include<stdio.h>

void main()

{

int var1= 60;

int var2= 1234;

scanf("%2d %5d", &var1, &var2);

}
```

Input data items should have to be separated by spaces, tabs or new-line and the punctuation marks are not counted as separators.

Reading and Writing Strings in C

There are two popular library functions *gets()* and *puts()* provides to deal with strings in C.
gets: The char *gets(char *str) reads a line from *stdin* and keeps the string pointed to by the *str* and is terminated when the new line is read or *EOF* is reached. The declaration of gets() function is:
Syntax:

char *gets(char *str);

Where str is a pointer to an array of characters where C strings are stored.
puts: The function - int puts(const char *str) is used to write a string to *stdout*, but it does not include null characters. A new line character needs to be appended to the output. The declaration is:
Syntax:

int puts(const char *str)

where str is the string to be written in C.