

Introduction To C

- In 1988, the American National Standards Institute (ANSI) had formalized the C language.
- C was invented to write UNIX operating system.
- C is a successor of 'Basic Combined Programming Language' (BCPL) called B language.
- Linux OS, PHP, and MySQL are written in C.
- C has been written in assembly language.

Uses of C Programming Language

In the beginning, C was used for developing system applications, e.g. :

- Database Systems
- Language Interpreters
- Compilers and Assemblers
- Operating Systems
- Network Drivers
- Word Processors

C Has Become Very Popular for Various Reasons

- One of the early programming languages.
- Still, the best programming language to learn quickly.
- C language is reliable, simple and easy to use.
- C language is a structured language.
- Modern programming concepts are based on C.
- It can be compiled on a variety of computer platforms.
- Universities preferred to add C programming in their courseware.

Features of C Programming Language

- C is a robust language with a rich set of built-in functions and operators.
- Programs written in C are efficient and fast.

- C is highly portable, programs once written in C can be run on other machines with minor or no modification.
- C is a collection of C library functions; we can also create our function and add it to the C library.
- C is easily extensible.

Advantages of C

- C is the building block for many other programming languages.
- Programs written in C are highly portable.
- Several standard functions are there (like in-built) that can be used to develop programs.
- C programs are collections of C library functions, and it's also easy to add own functions to the C library.
- The modular structure makes code debugging, maintenance and testing easier.

Disadvantages of C

- C does not provide Object Oriented Programming (OOP) concepts.
- There are no concepts of Namespace in C.
- C does not provide binding or wrapping up of data in a single unit.
- C does not provide Constructor and Destructor.

The limitations of C programming languages are as follows:

- Difficult to debug.
- C allows a lot of freedom in writing code, and that is why you can put an empty line or white space anywhere in the program. And because there is no fixed place to start or end the line, so it is difficult to read and understand the program.
- C compilers can only identify errors and are incapable of handling exceptions (run-time errors).
- C provides no data protection.
- It also doesn't feature reusability of source code extensively.
- It does not provide strict data type checking (for example an integer value can be passed for floating datatype).

C is imperative language and designed to compile in a relatively straightforward manner which provides low-level access to the memory. With the gradual

increase in the popularity of the program, the language and its compiler have become available on a wide range of platforms from embedded microcontrollers to supercomputers.

With the introduction of K&R C language (which is a new edition of C published in 1978 by Brian Kernighan and Denis Ritchie), several features have been included in C language.

Some of these features are:

- Standard I/O (Input/Output) Library
- long int - data type
- unsigned int - data type
- Compound assignment operators

During the late 1980's, C was started to use for a wide variety of mainframe computers, micro and mini computers which began to get famous. Gradually C got its superset - i.e., C++ which has added features, but its developed from C with all its initial concepts.

Compiler:

A compiler is a computer program that transforms human-readable (programming language) source code into another computer language (binary) code.

In simple terms, Compiler takes the code that you wrote and turned in to the binary code that computer can understand.

C compiler is a software application that transforms human-readable C program code to machine-readable code. The process of transforming the code from High-Level Language to Machine Level Language is called "Compilation". The human-readable code is the C program that consists of digits letters, special symbols, etc. which is understood by human beings. On the other hand, machine language is dependent on processor and processor understands zeroes and ones (binary) only. All C program execution is based on a processor which is available in the CPU; that is why entire C source code needs to be converted to the binary system by the compiler.

\

List Of C compilers:

Since there are various compilers available into the online market, here are the lists of some of the frequently used ones:

- CCS C Compiler
- Turbo C
- Minimalist GNU for Windows (MinGW)
- Portable C Compiler
- Clang C++
- Digital Mars C++ Compiler
- Intel C++
- IBM C++
- Visual C++ : Express Edition
- Oracle C++

All of these above compilers for C are free to download, but there are some other paid C compilers also available, or programmers can get it for trial version:

- Embarcadero C++
- Edison Design Group C++
- Green Hills C++
- HP C++ for Unix
- Intel C++ for Windows, Linux, and some embedded systems.
- Microsoft C++
- Paradigm C++

A C program involves the following sections:

- Documentations (Documentation Section)
- Preprocessor Statements (Link Section)
- Global Declarations (Definition Section)
- The main() function
 - Local Declarations

- Program Statements & Expressions
- User Defined Functions

Let's begin with a simple C program code.

Sample Code of C "Hello World" Program

Example:

```
/* Author: www.w3schools.in
Date: 2018-04-28
Description:
Writes the words "Hello, World!" on the screen */
#include<stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

or in a different way

```
/* Author: www.w3schools.in
Date: 2013-11-15
Description:
Writes the words "Hello, World!" on the screen */
```

```
#include<stdio.h>

#include<conio.h>

void main()

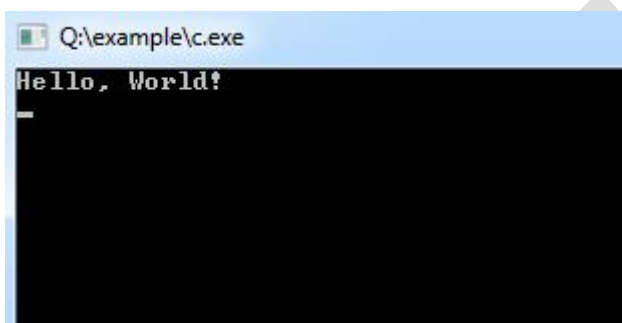
{

    printf("Hello, World!\n");

    return;

}
```

Program Output:



The above example has been used to print `Hello, World!` Text on the screen.

Let's look into various parts of the above C program.

`/* Comments */`

Comments are a way of explaining what makes a program. The compiler ignores comments and used by others to understand the code.

or

This is a comment block, which is ignored by the compiler. Comment can be used anywhere in the program to add info about program or code block, which will be helpful for developers to understand the existing code in the future easily.

PPS

<code>#include<stdio.h></code>	<p>stdio is standard for input / output, this allows us to use some commands which includes a file called stdio.h.</p> <p>or</p> <p>This is a preprocessor command. That notifies the compiler to include the header file stdio.h in the program before compiling the source-code.</p>
<code>int/void main()</code>	<p>int/void is a return value, which will be explained in a while.</p>
<code>main()</code>	<p>The main() is the main function where program execution begins. Every C program must contain only one main function.</p> <p>or</p> <p>This is the main function, which is the default entry point for every C program and the void in front of it indicates that it does not return a value.</p>
Braces	<p>Two curly brackets "{...}" are used to group all statements.</p> <p>or</p> <p>Curly braces which shows how much the main() function has its scope.</p>
<code>printf()</code>	<p>It is a function in C, which prints text on the screen.</p> <p>or</p> <p>This is another pre-defined function of C which is used to be displayed text string in the screen.</p>
<code>return 0</code>	<p>At the end of the main function returns value 0.</p>

Basic Structure of C Program:

The example discussed above illustrates how a simple C program looks like and how the program segment works. A C program may contain one or more sections which are figured above.

The Documentation section usually contains the collection of comment lines giving the name of the program, author's or programmer's name and few other details. The second part is the link-section which instructs the compiler to connect to the various functions from the system library. The Definition section describes all the symbolic-constants. The global declaration section is used to define those variables that are used globally within the entire program and is used in more than one function. This section also declares all the user-defined functions. Then comes the main(). All C programs must have a main() which contains two parts:

- Declaration part
- Execution part

The declaration part is used to declare all variables that will be used within the program. There needs to be at least one statement in the executable part, and these two parts are declared within the opening and closing curly braces of the main(). The execution of the program begins at the opening brace '{' and ends with the closing brace '}'. Also, it has to be noted that all the statements of these two parts need to be terminated with a semi-colon.

The sub-program section deals with all user-defined functions that are called from the main(). These user-defined functions are declared and usually defined after the main() function.

C input/output functions:

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.

```
#include<stdio.h>
void main()
```



```
{
int a,b,c;
printf("Please enter any two numbers: \n");
scanf("%d %d", &a, &b);
c = a + b;
printf("The addition of two number is: %d", c);
}
```

Output:

Please enter any two numbers:

12

3

The addition of two number is:15

The above program scanf() is used to take input from the user, and respectively printf() is used to display output result on the screen.

Managing Input/Output:

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, there are two important streams that play their role. These are:

- Standard Input (stdin)
- 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 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.

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

```
#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.

```
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);
```

```
#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.

```
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.

```
#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 Characters:

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.

C format specifiers:

Format specifiers can be defined as the operators which are used in association with `printf()` function for printing the data that is referred by any object or any variable. When a value is stored in a particular variable, then you cannot print the value stored in the variable straightforwardly without using the format specifiers. You can retrieve the data that are stored in the variables and can print them onto the console screen by implementing these format specifiers in a `printf()` function.

Format specifiers start with a percentage `%` operator and followed by a special character for identifying the type of the data.

There are mostly six types of format specifiers that are available in C.

List of format specifiers in C

Format specifier	Description
<code>%d</code>	Integer Format Specifier
<code>%f</code>	Float Format Specifier
<code>%c</code>	Character Format Specifier
<code>%s</code>	String Format Specifier
<code>%u</code>	Unsigned Integer Format Specifier
<code>%ld</code>	Long Int Format Specifier

Integer Format Specifier %d

The %d format specifier is implemented for representing integer values. This is used with printf() function for printing the integer value stored in the variable.

Syntax:

```
printf("%d",<variable name>);
```

Float Format Specifier %f

The %f format specifier is implemented for representing fractional values. This is implemented within printf() function for printing the fractional or floating value stored in the variable. Whenever you need to print any fractional or floating data, you have to use %f format specifier.

Syntax:

```
printf("%f", <variable name>);
```

Character Format Specifier %c

The %c format specifier is implemented for representing characters. This is used with printf() function for printing the character stored in a variable. When you want to print a character data, you should incorporate the %c format specifier.

Syntax:

```
printf("%c",<variable name>);
```

String Format Specifier %s

The %s format specifier is implemented for representing strings. This is used in printf() function for printing a string stored in the character array variable. When you have to print a string, you should implement the %s format specifier.

Syntax:

```
printf("%s",<variable name>);
```

Unsigned Integer Format Specifier %u

The %u format specifier is implemented for fetching values from the address of a variable having unsigned decimal integer stored in the memory. This is used within printf() function for printing the unsigned integer variable.

Syntax:

```
printf("%u",<variable name>);
```

Long Int Format Specifier %ld

The %ld format specifier is implemented for representing long integer values. This is implemented with printf() function for printing the long integer value stored in the variable.

Syntax:

```
printf("%ld",<variable name>);
```

In C programs, each individual word and punctuation is referred to as a token. C Tokens are the smallest building block or smallest unit of a C program.

C Supports Six Types of Tokens:

- Identifiers
- Keywords
- Constants
- Strings
- Operators
- Special Symbols

Identifiers are names given to different names given to entities such as constants, variables, structures, functions etc.

Example:

```
int amount;  
  
double totalbalance;
```

In the above example, amount and total balance are identifiers and int, and double are keywords.

Rules for Naming Identifiers

- An identifier can only have alphanumeric characters (a-z , A-Z , 0-9) (i.e. letters & digits) and underscore(_) symbol.
- Identifier names must be unique
- The first character must be an alphabet or underscore.
- You cannot use a keyword as identifiers.
- Only first thirty-one (31) characters are significant.
- Must not contain white spaces.
- Identifiers are case-sensitive.

C Keywords:

can't use a keyword as an identifier in your C programs, its reserved words in C library and used to perform an internal operation. The meaning and working of these keywords are already known to the compiler.

• C Keywords List

- A list of 32 reserved keywords in c language is given below:

auto	double	int	struct
------	--------	-----	--------

PPS

break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Example Where and How Keywords are Used in the Program

- Example:

```
• #include<stdio.h>
•
• main()
• {
• float a, b;
• printf("Showing how keywords are used.");
• return 0;
• }
```


- In the above program, `float` and `return` are keywords. The `float` is used to declare variables, and `return` is used to return an integer type value in this program.

Constants are like a variable, except that their value never changes during execution once defined.

C Constants is a most fundamental and essential part of C programming language. Constants in C are the fixed values that are used in a program, and its value remains the same during the entire execution of the program.

- Constants are also called literals.
- Constants can be any of the [data types](#).
- It is considered best practice to define constants using only `upper-case` names.

Constant Definition in C

Syntax:

```
const type constant_name;
```

`const` keyword defines a constant in C.

Example:

```
#include<stdio.h>

main()
{
    const int SIDE = 10;

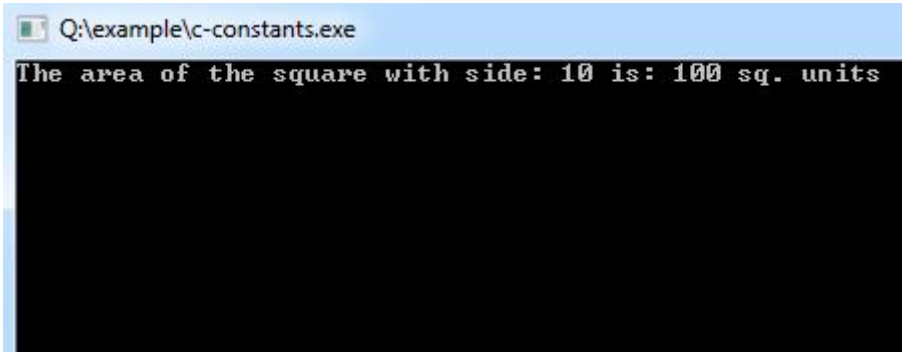
    int area;

    area = SIDE*SIDE;

    printf("The area of the square with side: %d is: %d sq. units"
, SIDE, area);
```

```
}
```

Program Output:



```
Q:\example\c-constants.exe
The area of the square with side: 10 is: 100 sq. units
```

Putting const either before or after the type is possible.

```
int const SIDE = 10;
```

or

```
const int SIDE = 10;
```

Constant Types in C

Constants are categorized into two basic types, and each of these types has own subtypes/categories. These are:

Primary Constants

1. Numeric Constants
 - Integer Constants
 - Real Constants
2. Character Constants
 - Single Character Constants
 - String Constants
 - Backslash Character Constants

Integer Constant

It's referring to a sequence of digits. Integers are of three types viz:

1. Decimal Integer
2. Octal Integer
3. Hexadecimal Integer

Example:

15, -265, 0, 99818, +25, 045, 0X6

Real constant

The numbers containing fractional parts like 99.25 are called real or floating points constant.

Single Character Constants

It simply contains a single character enclosed within ' and ' (a pair of single quote). It is to be noted that the character '8' is not the same as 8. Character constants have a specific set of integer values known as ASCII values (American Standard Code for Information Interchange).

Example:

'X', '5', ','

String Constants

These are a sequence of characters enclosed in double quotes, and they may include letters, digits, special characters, and blank spaces. It is again to be noted that "G" and 'G' are different - because "G" represents a string as it is enclosed within a pair of double quotes whereas 'G' represents a single character.

Example:

"Hello!", "2015", "2+1"

Backslash character constant

C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning which is known to the compiler. They are also termed as "Escape Sequence".

For Example:

`\t` is used to give a tab

`\n` is used to give a new line

Constants	Meaning
<code>\a</code>	beep sound
<code>\b</code>	backspace
<code>\f</code>	form feed
<code>\n</code>	new line
<code>\r</code>	carriage return
<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab
<code>\'</code>	single quote
<code>\"</code>	double quote

\\	backslash
\0	null

Secondary Constant

- [Array](#)
- [Pointer](#)
- [Structure](#)
- [Union](#)
- Enum

C Operators:

C operators are symbols that are used to perform mathematical or logical manipulations. The C programming language is rich with built-in operators. Operators take part in a program for manipulating data and variables and form a part of the mathematical or logical expressions.

C offers various types of operators having different functioning capabilities.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and Decrement Operators
- Conditional Operator
- Bitwise Operators
- Special Operators

Arithmetic Operators

Arithmetic Operators are used to performing mathematical calculations like addition (+), subtraction (-), multiplication (*), division (/) and modulus (%).

Operator	Description
----------	-------------

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

C Program to Add Two Numbers

Example:

```
#include <stdio.h>

void main()

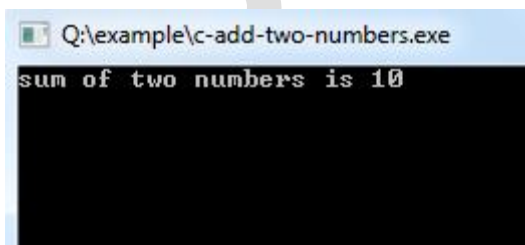
{

    int i=3,j=7,k; /* Variables Defining and Assign values */ k=i+j;

    printf("sum of two numbers is %d\n", k);

}
```

Program Output:



Increment and Decrement Operators

Increment and Decrement Operators are useful operators generally used to minimize the calculation, i.e. `++x` and `x++` means `x=x+1` or `-x` and `x--` means `x=x-1`. But there is a slight difference between `++` or `--` written before or after the operand. Applying the pre-increment first add one to the operand and then the result is assigned to the variable on the left whereas post-increment first assigns the value to the variable on the left and then increment the operand.

Operator	Description
<code>++</code>	Increment
<code>--</code>	Decrement

Example: To Demonstrate prefix and postfix modes.

```
#include <stdio.h>

//stdio.h is a header file used for input.output purpose.

void main()
{
    //set a and b both equal to 5.
    int a=5, b=5;

    //Print them and decrementing each time.

    //Use postfix mode for a and prefix mode for b.

    printf("\n%d %d",a--,--b);

    printf("\n%d %d",a--,--b);

    printf("\n%d %d",a--,--b);

    printf("\n%d %d",a--,--b);
```

```
printf("\n%d %d", a--, --b);
}
```

Program Output:

```
5 4
4 3
3 2
2 1
1 0
```

Relational Operators

Relational operators are used to compare two quantities or values.

Operator	Description
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Logical Operators

C provides three logical operators when we test more than one condition to make decisions. These are: `&&` (meaning logical AND), `||` (meaning logical OR) and `!` (meaning logical NOT).

Operator	Description
<code>&&</code>	And operator. It performs a logical conjunction of two expressions. (if both expressions evaluate to True, result is True. If either expression evaluates to False, the result is False)
<code> </code>	Or operator. It performs a logical disjunction on two expressions. (if either or both expressions evaluate to True, the result is True)
<code>!</code>	Not operator. It performs logical negation on an expression.

Bitwise Operators

C provides a special operator for bit operation between two variables.

Operator	Description
<code><<</code>	Binary Left Shift Operator
<code>>></code>	Binary Right Shift Operator
<code>~</code>	Binary One's Complement Operator
<code>&</code>	Binary AND Operator
<code>^</code>	Binary XOR Operator

	Binary OR Operator
--	--------------------

Assignment Operators

Assignment operators applied to assign the result of an expression to a variable. C has a collection of shorthand assignment operators.

Operator	Description
=	Assign
+=	Increments then assign
-=	Decrements then assign
*=	Multiplies then assign
/=	Divides then assign
%=	Modulus then assign
<<=	Left shift and assign
>>=	Right shift and assign
&=	Bitwise AND assign
^=	Bitwise exclusive OR and assign

=	Bitwise inclusive OR and assign
---	---------------------------------

Conditional Operator

C offers a ternary operator which is the conditional operator (?: in combination) to construct conditional expressions.

Operator	Description
?:	Conditional Expression

Special Operators

C supports some special operators

Operator	Description
sizeof()	Returns the size of a memory location.
&	Returns the address of a memory location.
*	Pointer to a variable.

Program to demonstrate the use of sizeof operator

Example:

```
#include <stdio.h>

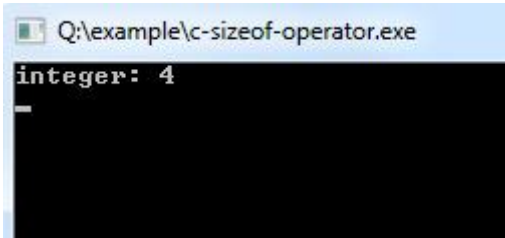
void main()

{
```

```
int i=10; /* Variables Defining and Assign values */
printf("integer: %d\n", sizeof(i));

}
```

Program Output:



```
Q:\example\c-sizeof-operator.exe
integer: 4
```

C Data Types:

A data-type in C programming is a set of values and is determined to act on those values. C provides various types of data-types which allow the programmer to select the appropriate type for the variable to set its value.

The data-type in a programming language is the collection of data with values having fixed meaning as well as characteristics. Some of them are an integer, floating point, character, etc. Usually, programming languages specify the range values for given data-type.

C Data Types are used to:

- Identify the type of a variable when it declared.
- Identify the type of the return value of a function.
- Identify the type of a parameter expected by a function.

ANSI C provides three types of data types:

1. Primary(Built-in) Data Types:
void, int, char, double and float.
2. Derived Data Types:
Array, References, and Pointers.
3. User Defined Data Types:
Structure, Union, and Enumeration.

Primary Data Types

Every C compiler supports five primary data types:

void	As the name suggests it holds no value and is generally used for specifying the return type of function or what it returns. If the function has a void type, it means that the function will not return any value.
int	Used to denote an integer type.
char	Used to denote a character type.
float, double	Used to denote a floating point type.
int *, float *, char *	Used to denote a pointer type.

Three more data types have been added in C99:

- `_Bool`
- `_Complex`
- `_Imaginary`

Declaration of Primary Data Types with Variable Names

After taking suitable variable names, they need to be assigned with a data type. This is how the data types are used along with variables:

Example:

```
int    age;

char   letter;

float  height, width;
```

Derived Data Types

C supports three derived data types:

Data Types	Description
Arrays	Arrays are sequences of data items having homogeneous values. They have adjacent memory locations to store values.
References	Function pointers allow referencing functions with a particular signature.
Pointers	These are powerful C features which are used to access the memory and deal with their addresses.

User Defined Data Types

C allows the feature called `type definition` which allows programmers to define their identifier that would represent an existing data type. There are three such types:

Data Types	Description
Structure	It is a package of variables of different types under a single name. This is done to handle data efficiently. "struct" keyword is used to define a structure.
Union	These allow storing various data types in the same memory location. Programmers can define a union with different members, but only a single member can contain a value at given time. It is used for
Enum	Enumeration is a special data type that consists of integral constants, and each of them is assigned with a specific name. "enum" keyword is used to define the enumerated data type.

Data Types and Variable Declarations in C

Example:

```
#include <stdio.h>

int main()
{
    int a = 4000; // positive integer data type

    float b = 5.2324; // float data type

    char c = 'z'; // char data type

    long d = 41657; // long positive integer data type

    long e = -21556; // long -ve integer data type

    int f = -185; // -ve integer data type

    short g = 130; // short +ve integer data type

    short h = -130; // short -ve integer data type

    double i = 4.1234567890; // double float data type

    float j = -3.55; // float data type
}
```

The storage representation and machine instructions differ from machine to machine. `sizeof` operator can use to get the exact size of a type or a variable on a particular platform.

Example:

```
#include <stdio.h>

#include <limits.h>

int main()
{
```

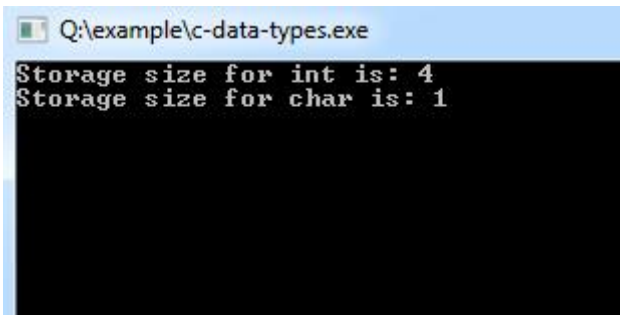
```
printf("Storage size for int is: %d \n", sizeof(int));

printf("Storage size for char is: %d \n", sizeof(char));

return 0;

}
```

Program Output:



```
Q:\example\c-data-types.exe
Storage size for int is: 4
Storage size for char is: 1
```

C Variables:

Variables are memory locations(storage area) in C programming language.

The primary purpose of variables is to store data in memory for later use.

Unlike [constants](#) which do not change during the program execution, variables value may change during execution. If you declare a variable in C, that means you are asking to the operating system for reserve a piece of memory with that variable name.

Variable Definition in C

Syntax:

```
type variable_name;
```

or

```
type variable_name, variable_name, variable_name;
```

Variable Definition and Initialization

Example:

```
int    width, height=5;

char   letter='A';

float  age, area;

double d;

/* actual initialization */width = 10;

age = 26.5;
```

Variable Assignment

Variable assignment is a process of assigning a value to a variable.

Example:

```
int width = 60;

int age = 31;
```

There are some rules on choosing variable names

- A variable name can consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and the underscore character.
- The first character must be a letter or underscore.
- Blank spaces cannot be used in variable names.
- Special characters like #, \$ are not allowed.
- C keywords cannot be used as variable names.
- Variable names are case sensitive.
- Values of the variables can be numeric or alphabetic.
- Variable type can be char, int, float, double or void.

C Program to Print Value of a Variable

Example:

```
#include<stdio.h>

void main()

{

    /* c program to print value of a variable */    int age = 33;

    printf("I am %d years old.\n", age);

}
```

Program Output:

```
I am 33 years old.
```

Storage Classes:

Storage Classes are associated with variables for describing the features of any variable or function in C program. These storage classes deal with features such as scope, lifetime and visibility which helps programmers to define a particular variable during program's runtime. These storage classes are preceded by the data type which they had to modify.

There are four storage classes types in C:

- auto
- register
- static
- extern

auto Storage Class

auto comes by default with all local variables as its storage class. The keyword auto is used to define this storage class explicitly

Syntax:

```
int roll; // contains auto by default
```

is same as:

```
auto int roll; // in addition, we can use auto keyword
```

The above example has a variable name `roll` with `auto` as a storage class. This storage class can only be implemented with the local variables.

register Storage Class

This storage class is implemented for classifying local variables whose value needs to be saved in a register in place of RAM (Random Access Memory). This is implemented when you want your variable the maximum size equivalent to the size of register. It uses the keyword `register`.

Syntax:

```
register int counter;
```

Register variables are used when implementing looping in counter variables to make program execution fast. Register variables work faster than variables stored in RAM (primary memory).

Example:

```
for(register int counter=0; counter<=9; counter++)  
{  
    // loop body  
}
```

static storage class

This storage class uses static variables that are used popularly for writing programs in C language. Static variables preserve the value of a variable even when the scope limit exceeds. Static storage class has its scope local to the function in which it is defined. On the other hand, global static variables can be accessed in any part of your program. The default value assigned is '0' by the C compiler. The keyword used to define this storage class is `static`.

Example:

```
static int var = 6;
```

extern Storage class

The extern storage class is used to feature a variable to be used from within different blocks of the same program. Mainly, a value is set to that variable which is in a different block or function and can be overwritten or altered from within another block as well. Hence it can be said that an extern variable is a global variable which is assigned with a value that can be accessed and used within the entire program. Moreover, a global variable can be explicitly made an extern variable by implementing the keyword 'extern' preceded the variable name.

Here are some examples of extern:

Example:

```
#include <stdio.h>

int val;

extern void funcExtern();

main()
```

```
{  
  
    val = 10;  
  
    funcExtern();  
  
}
```

Another example:

Example:

```
#include <stdio.h>  
  
extern int val; // now the variable val can be accessed and used from  
anywhere  
  
// within the program  
void funcExtern()  
{  
  
    printf("Value is: %d\n", val);  
  
}
```

C Preprocessors:

The preprocessor is a program invoked by the compiler that modifies the source code before the actual compilation takes place.

To use any preprocessor directives, first, we have to prefix them with pound symbol #.

The following section lists all preprocessor directives:

Category	Directive	Description
Macro substitution division	#include	File include
	#define #undef	Macro define, Macro undefine
	#ifdef #ifndef	If macro defined, If macro not defined
File inclusion division	#if #elif #else #endif	If, Else, ifElse, End if
Compiler control division	#line #error #pragma	Set line number, Abort compilation, Set compiler option

C Preprocessors Examples

Syntax:

```
#include <stdio.h>

/* #define macro_name character_sequence */

#define LIMIT 10

int main()
{
    int counter;
```

```
for(counter =1; counter <=LIMIT; counter++)  
  
{  
  
    printf("%d\n",counter);  
  
}  
  
return 0;  
  
}
```

In above example for loop will run 10 times.

```
#include <stdio.h>  
  
#include "header.h"
```

`#include <stdio.h>` tell the compiler to add `stdio.h` file from System Libraries to the current source file, and `#include "header.h"` tells compiler to get `header.h` from the local directory.

```
#undef LIMIT  
  
#define LIMIT 20
```

This tells the compiler to undefine existing `LIMIT` and set it as 20.

```
#ifndef LIMIT  
  
    #define LIMIT 50  
  
#endif
```

This tells the compiler to define `LIMIT`, only if `LIMIT` isn't already defined.

```
#ifdef LIMIT  
  
    /* Your statements here */#endif
```

C Header Files:

C language is famous for its different libraries and the predefined functions pre-written within it. These make programmer's effort a lot easier. In this tutorial, you will be learning about C header files and how these header files can be included in your C program and how it works within your C language.

Header files are helping file of your C program which holds the definitions of various functions and their associated variables that needs to be imported into your C program with the help of pre-processor `#include` statement. All the header file have a '.h' an extension that contains C function declaration and macro definitions. In other words, the header files can be requested using the preprocessor directive `#include`. The default header file that comes with the C compiler is the `stdio.h`.

Including a header file means that using the content of header file in your source program. A straightforward practice while programming in C or C++ programs is that you can keep every macro, global variables, constants, and other function prototypes in the header files. The basic syntax of using these header files is:

Syntax:

```
#include <file>
```

or

```
#include "file"
```

This kind of file inclusion is implemented for including system oriented header files. This technique (with angular braces) searches for your file-name in the standard list of system directories or within the compiler's directory of header files. Whereas, the second kind of header file is used for user defined header files or other external files for your program. This technique is used to search for the file(s) within the directory that contains the current file.

How include Works

The C's #include preprocessor directive statement exerts by going through the C preprocessors for scanning any specific file like that of input before abiding by the rest of your existing source file. Let us take an example where you may think of having a header file `karl.h` having the following statement:

Example:

```
char *example (void);
```

then, you have a main C source program which seems something like this:

Example:

```
#include<stdio.h>

int x;

#include "karl.h"

int main ()
{
    printf("Program done");
    return 0;
}
```

So, the compiler will see the entire C program and token stream as:

Example:

```
#include<stdio.h>

int x;

char * example (void);
```

```
int main ()  
  
{  
  
    printf("Program done");  
  
    return 0;  
  
}
```

Writing of Single and Multiple uses of Header files

You can use various header files based on some conditions. In case, when a header file needs to be included twice within your program, your compiler will be going to process the contents inside it - twice which will eventually lead to an error in your program. So to eliminate this, you have to use conditional preprocessor directives. Here's the syntax:

Syntax:

```
#ifndef HEADER_FILE_NAME  
  
#define HEADER_FILE_NAME  
  
    the entire header file  
  
#endif
```

Again, sometimes it's essential for selecting several diverse header files based on some requirement to be incorporated into your program. For this also multiple conditional preprocessors can be used like this:

Syntax:

```
#if FIRST_SYSTEM  
  
    #include "sys.h"  
  
#elif SEC_SYSTEM  
  
    #include "sys2.h"
```

```
#elif THRID_SYSTEM  
  
.....  
  
#endif
```

C Type Casting:

Type Casting in C is used to convert a variable from one data type to another data type, and after type casting compiler treats the variable as of the new data type.

Syntax:

```
(type_name) expression
```

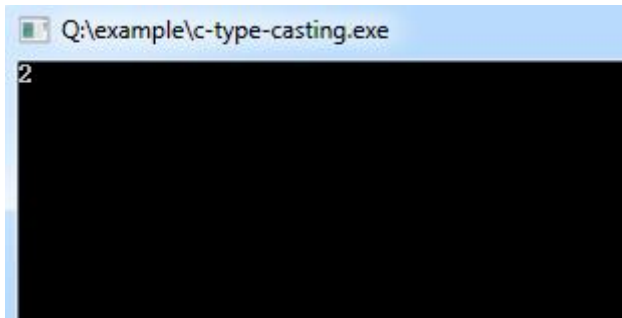
Without Type Casting

Example:

```
#include <stdio.h>  
  
main ()  
{  
  
    int a;  
  
    a = 15/6;  
  
    printf("%d",a);  
  
}
```

Program Output:

In the above C program, 15/6 alone will produce integer value as 2.



After Type Casting

```
#include <stdio.h>

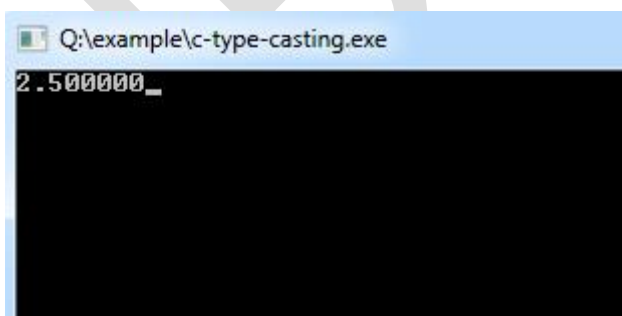
main ()
{
    float a;

    a = (float) 15/6;

    printf("%f", a);
}
```

Program Output:

After type cast is done before division to retain float value 2.500000.



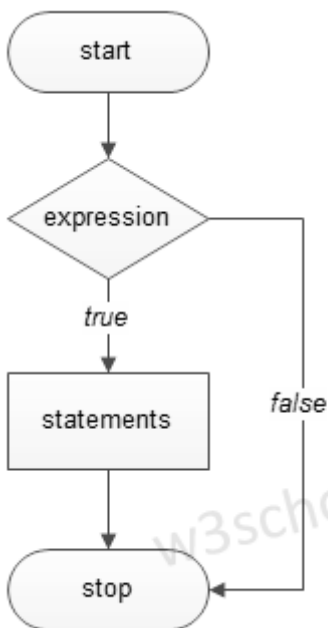
C Conditional Statements:

C conditional statements allow you to make a decision, based upon the result of a condition. These statements are called as Decision Making Statements or Conditional Statements.

So far, we have seen that all set of statements in a C program gets executed sequentially in the order in which they are written and appear. This occurs when there is no jump based statements or repetitions of certain calculations. But some situations may arise where we may have to change the order of execution of statements depending on some specific conditions. This involves a kind of decision making from a set of calculations. It is to be noted that C language assumes any non-zero or non-null value as true and if zero or null, treated as false.

This type of structure requires that the programmers indicate several conditions for evaluation within a program. The statement(s) will get executed only if the condition becomes true and optionally, alternative statement or set of statements will get executed if the condition becomes false.

The flowchart of Decision-making technique in C can be expressed as:



C languages have such decision-making capabilities within its program by the use of following the decision making statements:

Conditional Statements in C

- If statement
 - [if statement](#)

- if-else statement
- Nested if-else statement
- else if-statement
- goto statement
- switch statement
- Conditional Operator

C If statements:

If statements in C is used to control the program flow based on some condition, it's used to execute some statement code block if the expression is evaluated to true. Otherwise, it will get skipped. This is the simplest way to modify the control flow of the program.

The if statement in C can be used in various forms depending on the situation and complexity.

There are four different types of if statement in C. These are:

- Simple if Statement
- if-else Statement
- Nested if-else Statement
- else-if Ladder

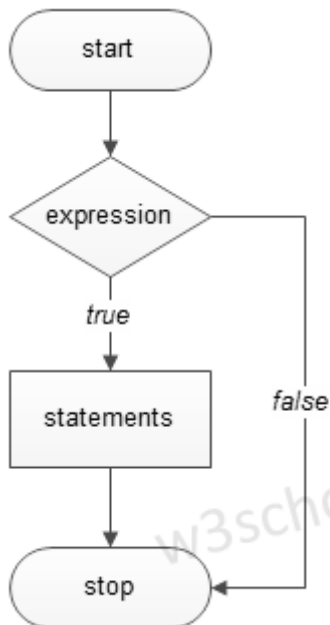
The basic format of if statement is:

Syntax:

```
if(test_expression)
{
    statement 1;
    statement 2;
    ...
}
```

'Statement n' can be a statement or a set of statements, and if the test expression is evaluated to `true`, the statement block will get executed, or it will get skipped.

Figure - Flowchart of if Statement:



Example of a C Program to Demonstrate if Statement

Example:

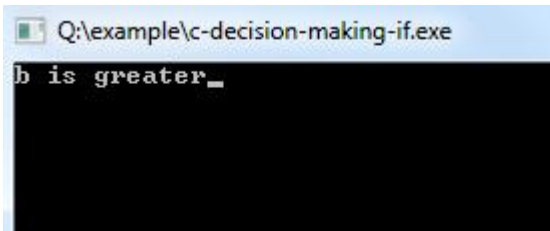
```
#include<stdio.h>

main ()
{
    int a = 15, b = 20;

    if (b > a) {
        printf("b is greater");
    }
}
```

```
}
```

Program Output:



```
Q:\example\c-decision-making-if.exe
b is greater_
```

Example:

```
#include<stdio.h>

main()
{
    int number;

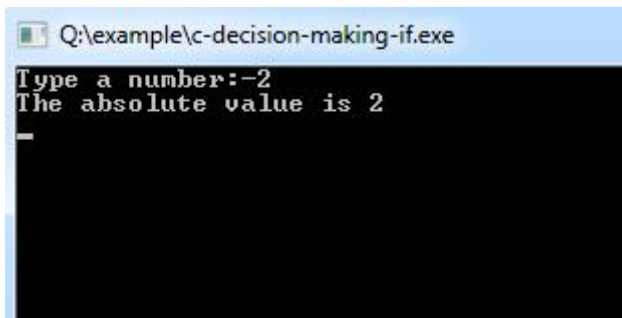
    printf( & quot; Type a number: & quot;);

    scanf( & quot; % d & quot;, & amp; number);

    /* check whether the number is negative number */ if (number & lt; 0)
    {
        /* If it is a negative then convert it into positive. */
        number = -number;

        printf( & quot; The absolute value is % d\ n & quot;, number);
    }

    getch();
}
```


Program Output:

```
Q:\example\c-decision-making-if.exe
Type a number:-2
The absolute value is 2
-
```

If-Else Statements:

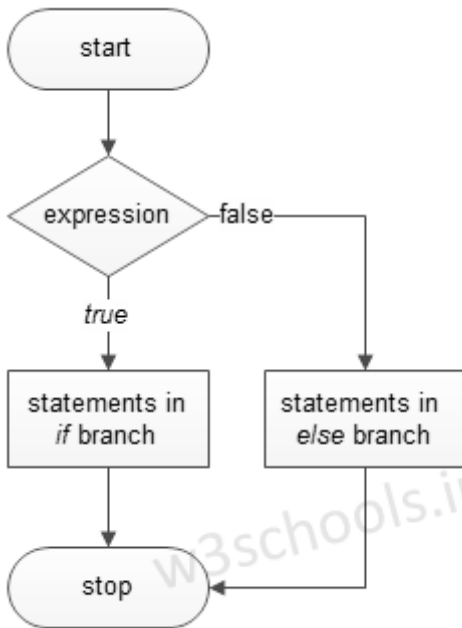
If else statements in C is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if the expression is evaluated to true, otherwise executes else statement code block.

The basic format of if else statement is:

Syntax:

```
if(test_expression)
{
    //execute your code
}
else
{
    //execute your code
}
```

Figure - Flowchart of if-else Statement:



Example of a C Program to Demonstrate if-else Statement

Example:

```
#include<stdio.h>

main()
{
    int a, b;

    printf("Please enter the value for a:");

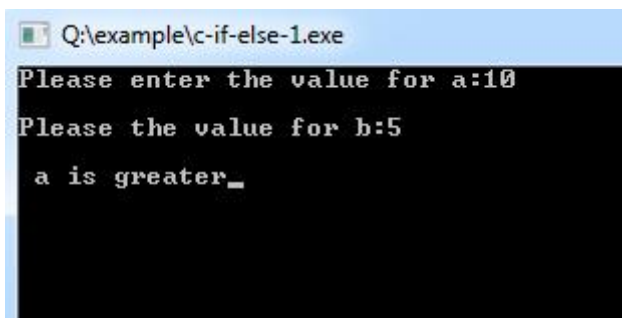
    scanf("%d", & a);

    printf("\nPlease the value for b:");

    scanf("%d", & b);
```

```
if (a > b) {  
    printf("\n a is greater");  
} else {  
    printf("\n b is greater");  
}  
}
```

Program Output:



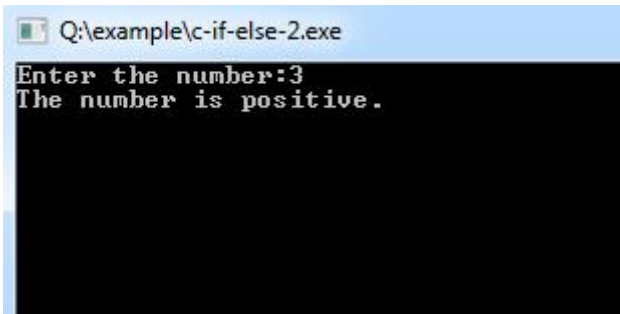
```
Q:\example\c-if-else-1.exe  
Please enter the value for a:10  
Please the value for b:5  
a is greater_
```

Example:

```
#include<stdio.h>  
  
main() {  
    int num;  
    printf("Enter the number:");  
    scanf("%d", num);  
  
    /* check whether the number is negative number */ if (num < 0)  
        printf("The number is negative.");  
    else  
        printf("The number is positive.");  
}
```

```
}
```

Program Output:



```
Q:\example\c-if-else-2.exe
Enter the number:3
The number is positive.
```

Nested If-Else Statements:

Nested if else statements play an important role in C programming, it means you can use conditional statements inside another conditional statement.

The basic format of Nested if else statement is:

Syntax:

```
if(test_expression one)
{
    if(test_expression two) {
        //Statement block Executes when the boolean test expression two is
true.
    }
}
else
{
    //else statement block
}
```

Example of a C Program to Demonstrate Nested if-else Statement

Example:

```
#include<stdio.h>

main()
{
int x=20,y=30;

if(x==20)
{
if(y==30)
{
printf("value of x is 20, and value of y is 30.");
}
}
}
```

Execution of the above code produces the following result.

Output:

```
value of x is 20, and value of y is 30.
```

Else-if statements:

else-if statements in C is like another if condition, it's used in a program when if statement having multiple decisions.

The basic format of else if statement is:

Syntax:

```
if(test_expression)
{
    //execute your code
}
else if(test_expression n)
{
    //execute your code
}
else
{
    //execute your code
}
```

Example of a C Program to Demonstrate else if Statement

Example:

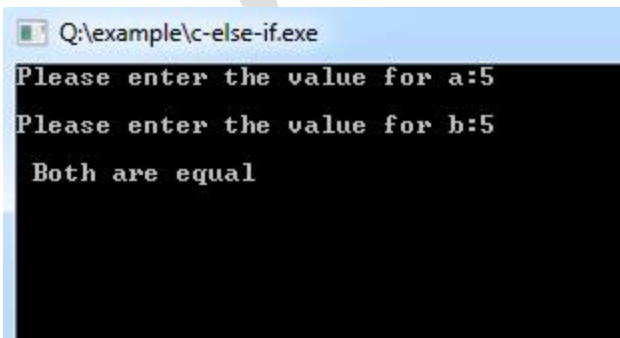
```
#include<stdio.h>

main()
{
    int a, b;

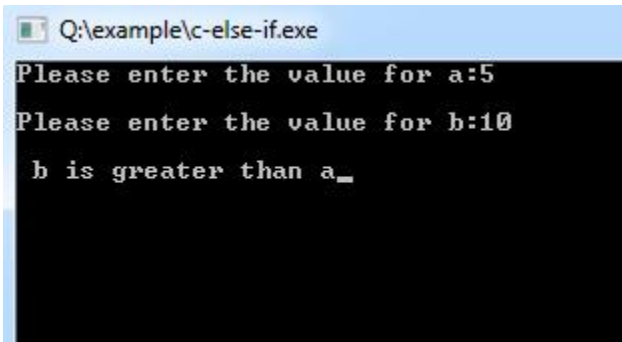
    printf("Please enter the value for a:");
    scanf("%d", & a);
```

```
printf("\nPlease enter the value for b:");  
  
scanf("%d", & b);  
  
if (a > b)  
{  
    printf("\n a is greater than b");  
}  
  
else if (b > a)  
{  
    printf("\n b is greater than a");  
}  
  
else  
{  
    printf("\n Both are equal");  
}  
}
```

Program Output:



```
Q:\example\c-else-if.exe  
Please enter the value for a:5  
Please enter the value for b:5  
Both are equal
```



```
Q:\example\c-else-if.exe
Please enter the value for a:5
Please enter the value for b:10
b is greater than a_
```

Goto statements:

So far we have discussed the if statements and how it is used in C to control statement execution based on some decisions or conditions. The flow of execution also depends on other statements which are not based on conditions that can control the flow.

C supports a unique form of a statement that is the goto Statement which is used to branch unconditionally within a program from one point to another. Although it is not a good habit to use goto statement in C, there may be some situations where the use of goto statement might be desirable.

The goto statement is used by programmers to change the sequence of execution of a C program by shifting the control to a different part of the same program.

The general form of the goto statement is:

Syntax:

```
goto label;
```

A label is an identifier required for goto statement to a place where the branch is to be made. A label is a valid variable name which is followed by a colon and is put immediately before the statement where the control needs to be jumped/transferred unconditionally.

Syntax:

```
goto label;
```



```
-----  
-----
```

```
label:
```

```
statement - X;
```

```
/* This the forward jump of goto statement */
```

or

```
label:
```

```
-----  
-----
```

```
goto label;
```

```
/*This is the backward jump of goto statement */
```

An Example of a C Program to Demonstrate goto Statement

Example:

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
int age;

g: //label name

    printf("you are Eligible\n");

s: //label name

    printf("you are not Eligible");

printf("Enter you age:");

scanf("%d", &age);

if(age>=18)

    goto g; //goto label g

else

    goto s; //goto label s

getch();

}
```

Switch statements:

C switch statement is used when you have multiple possibilities for the if statement.

The basic format of the switch statement is:

Syntax:

```
switch (variable)

{

case 1:

    //execute your code
```

```
break;

case n:

    //execute your code

break;

default:

    //execute your code

break;

}
```

After the end of each block it is necessary to insert a `break` statement because if the programmers do not use the `break` statement, all consecutive blocks of codes will get executed from every case onwards after matching the case block.

Example of a C Program to Demonstrate Switch Statement

Example:

```
#include<stdio.h>

main()

{

    int a;

    printf("Please enter a no between 1 and 5: ");

    scanf("%d",&a);
```

```
switch(a)
{
case 1:
printf("You chose One");
break;

case 2:
printf("You chose Two");
break;

case 3:
printf("You chose Three");
break;

case 4:
printf("You chose Four");
break;

case 5:
printf("You chose Five.");
break;

default :

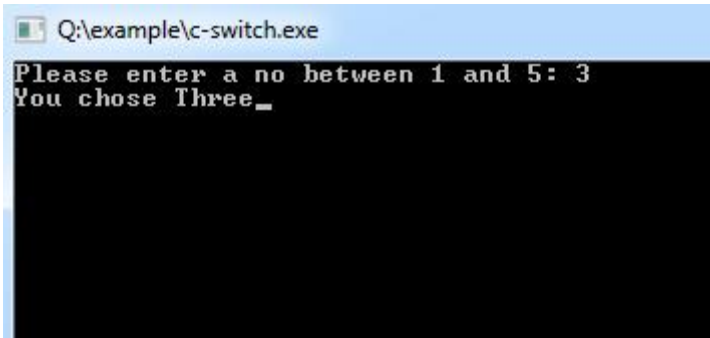
printf("Invalid Choice. Enter a no between 1 and 5");
```

```
break;
```

```
}
```

```
}
```

Program Output:



```
Q:\example\c-switch.exe
Please enter a no between 1 and 5: 3
You chose Three_
```

When none of the cases is evaluated to true, the default case will be executed, and break statement is not required for default statement.

Loops:

Sometimes it is necessary for the program to execute the statement several times, and C loops execute a block of commands a specified number of times until a condition is met. In this chapter, you will learn about all the looping statements of C programming along with their use.

A computer is the most suitable machine to perform repetitive tasks and can tirelessly do a task tens of thousands of times. Every programming language has the feature to instruct to do such repetitive tasks with the help of certain form of statements. The process of repeatedly executing a collection of statement is called looping. The statements get executed many numbers of times based on the condition. But if the condition is given in such logic that the repetition continues any number of times with no fixed condition to stop looping those statements, then this type of looping is called infinite looping.

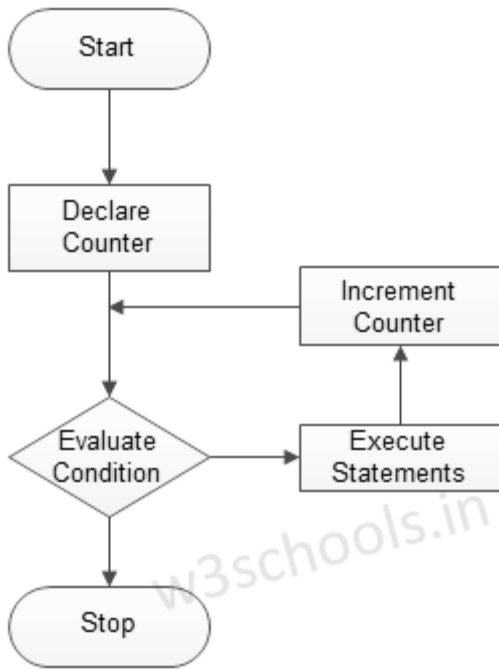
C supports following types of loops:

- [while loops](#)
- [do while loops](#)

- [for loops](#)

All are slightly different and provides loops for different situations.

Figure - Flowchart of Looping:



C Loop Control Statements

Loop control statements are used to change the normal sequence of execution of the loop.

Statement	Syntax	Description
break statement	break;	Is used to terminate loop or switch statements.
continue statement	continue;	Is used to suspend the execution of current loop iteration and transfer control to the loop for the next iteration.
goto statement	goto labelName; labelName: statement;	It transfers current program execution sequence to some other part of the program.

While loops:

C while loops statement allows to repeatedly run the same block of code until a condition is met.

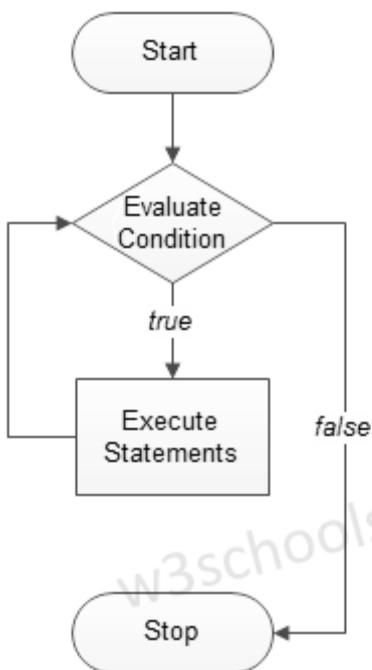
while loop is a most basic loop in C programming. while loop has one control condition, and executes as long the condition is true. The condition of the loop is tested before the body of the loop is executed, hence it is called an entry-controlled loop.

The basic format of while loop statement is:

Syntax:

```
While (condition)
{
    statement (s);
    Incrementation;
}
```

Figure - Flowchart of while loop:



Example of a C Program to Demonstrate while loop

Example:

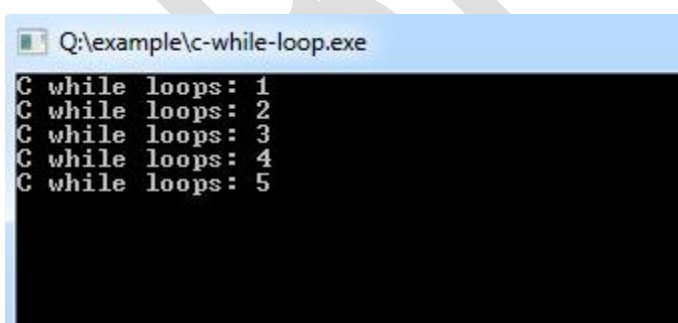
```
#include<stdio.h>

int main ()
{
    /* local variable Initialization */    int n = 1,times=5;

    /* while loops execution */    while( n <= times )
    {
        printf("C while loops: %d\n", n);
        n++;
    }

    return 0;
}
```

Program Output:



```
Q:\example\c-while-loop.exe
C while loops: 1
C while loops: 2
C while loops: 3
C while loops: 4
C while loops: 5
```

Do while loops:

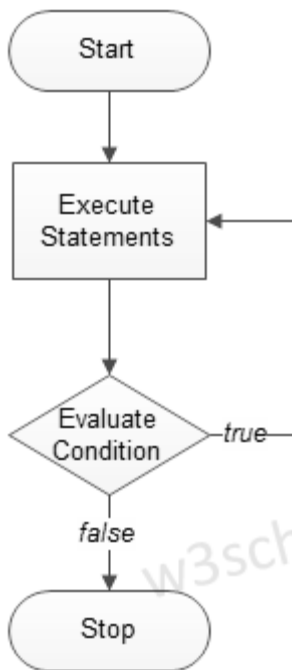
C do while loops are very similar to the while loops, but it always executes the code block at least once and furthermore as long as the condition remains true. This is an exit-controlled loop.

The basic format of do while loop statement is:

Syntax:

```
do
{
    statement (s) ;
}while ( condition ) ;
```

Figure - Flowchart of do while loop:



Example of a C Program to Demonstrate do while loop

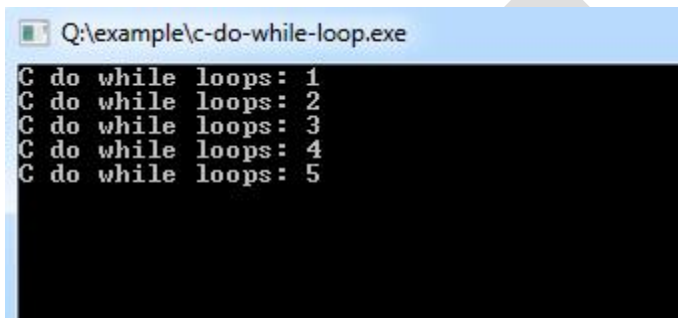
Example:

```
#include<stdio.h>

int main ()
```

```
{  
    /* local variable Initialization */    int n = 1,times=5;  
  
    /* do loops execution */    do  
  
    {  
  
        printf("C do while loops: %d\n", n);  
  
        n = n + 1;  
  
    }while( n <= times );  
  
    return 0;  
}
```

Program Output:



```
Q:\example\c-do-while-loop.exe  
C do while loops: 1  
C do while loops: 2  
C do while loops: 3  
C do while loops: 4  
C do while loops: 5
```

For loops:

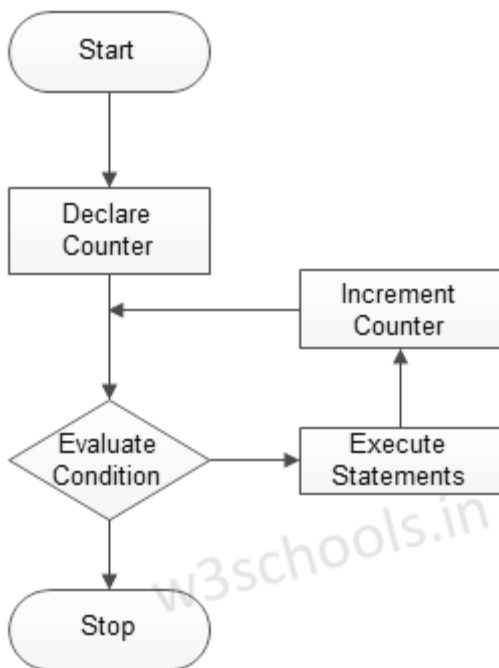
C for loops is very similar to a while loops in that it continues to process a block of code until a statement becomes false, and everything is defined in a single line. The for loop is also entry-controlled loop.

The basic format of for loop statement is:

Syntax:

```
for ( init; condition; increment )  
  
{  
  
    statement (s);  
  
}
```

Figure - Flowchart of for loop:



Example of a C Program to Demonstrate for loop

Example:

```

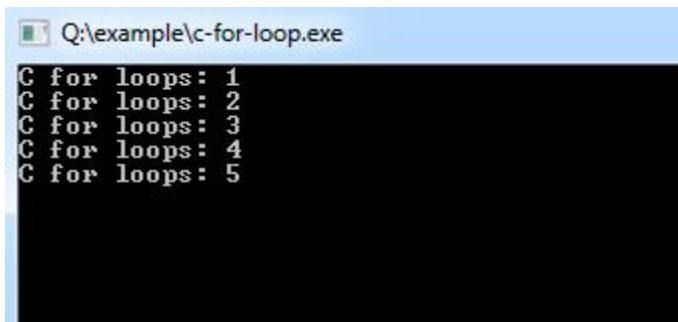
#include<stdio.h>

int main ()
{
  /* local variable Initialization */  int n,times=5;;

  /* for loops execution */  for( n = 1; n <= times; n = n + 1 )
  {
    printf("C for loops: %d\n", n);
  }

  return 0;
}
  
```

Program Output:



```
Q:\example\c-for-loop.exe
C for loops : 1
C for loops : 2
C for loops : 3
C for loops : 4
C for loops : 5
```

Functions:

C function is a self-contained block of statements that can be executed repeatedly whenever we need it.

Benefits of using function in C

- The function provides modularity.
- The function provides reusable code.
- In large programs, debugging and editing tasks is easy with the use of functions.
- The program can be modularized into smaller parts.
- Separate function independently can be developed according to the needs.

There are two types of functions in C

- **Built-in(Library) Functions**
 - The system provided these functions and stored in the library. Therefore it is also called Library Functions.
e.g. `scanf()`, `printf()`, `strcpy`, `strlwr`, `strcmp`, `strlen`, `strcat` etc.
 - To use these functions, you just need to include the appropriate C header files.
- **User Defined Functions** These functions are defined by the user at the time of writing the program.

Parts of Function

1. Function Prototype (function declaration)
2. Function Definition

3. Function Call

1. Function Prototype

Syntax:

```
dataType functionName (Parameter List)
```

Example:

```
int addition();
```

2. Function Definition

Syntax:

```
returnType functionName(Function arguments){  
  
    //body of the function  
  
}
```

Example:

```
int addition()  
{  
  
}
```

3. Calling a function in C

Program to illustrate Addition of Two Numbers using User Defined Function

Example:

```
#include<stdio.h>
```

```
/* function declaration */int addition();

int main()
{
    /* local variable definition */    int answer;

    /* calling a function to get addition value */    answer =
addition();

    printf("The addition of two numbers is: %d\n",answer);

    return 0;
}

/* function returning the addition of two numbers */int addition()
{
    /* local variable definition */    int num1 = 10, num2 = 5;

    return num1+num2;
}
```

Program Output:

The addition of two numbers is: 15

Function arguments:

While calling a function, the arguments can be passed to a function in two ways, Call by value and call by reference.

Type	Description
------	-------------

Call by Value	<ul style="list-style-type: none">• The actual parameter is passed to a function.• New memory area created for the passed parameters, can be used only within the function.• The actual parameters cannot be modified here.
Call by Reference	<ul style="list-style-type: none">• Instead of copying variable; an address is passed to function as parameters.• Address operator(&) is used in the parameter of the called function.• Changes in function reflect the change of the original variables.

Call by Value

Example:

```
#include<stdio.h>

/* function declaration */int addition(int num1, int num2);

int main()
{
    /* local variable definition */    int answer;

    int num1 = 10;
    int num2 = 5;

    /* calling a function to get addition value */    answer =
    addition(num1,num2);

    printf("The addition of two numbers is: %d\n",answer);

    return 0;
}
```

```
/* function returning the addition of two numbers */int addition(int
a,int b)

{

    return a + b;

}
```

Program Output:

The addition of two numbers is: 15

Call by Reference

Example:

```
#include<stdio.h>

/* function declaration */int addition(int *num1, int *num2);

int main()
{

    /* local variable definition */    int answer;

    int num1 = 10;

    int num2 = 5;

    /* calling a function to get addition value */    answer =
addition(&num1,&num2);

    printf("The addition of two numbers is: %d\n",answer);

    return 0;
```



```
}

/* function returning the addition of two numbers */int addition(int
*a,int *b)

{

    return *a + *b;

}
```

Program Output:

The addition of two numbers is: 15

Library functions:

The C library functions are provided by the system and stored in the library. The C library function is also called an inbuilt function in C programming.

To use Inbuilt Function in C, you must include their respective header files, which contain prototypes and data definitions of the function.

C Program to Demonstrate the Use of Library Functions

Example:

```
#include<stdio.h>

#include<ctype.h>

#include<math.h>

void main ()

{

    int i = -10, e = 2, d = 10; /* Variables Defining and Assign values */
    float rad = 1.43;
```

```
double d1 = 3.0, d2 = 4.0;

printf("%d\n", abs(i));

printf("%f\n", sin(rad));

printf("%f\n", cos(rad));

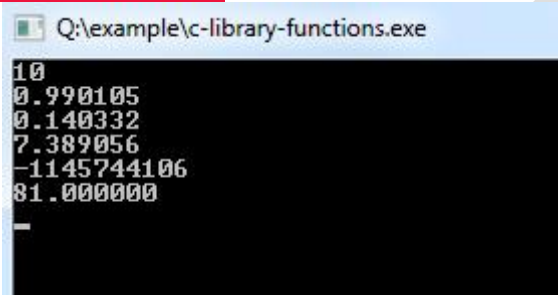
printf("%f\n", exp(e));

printf("%d\n", log(d));

printf("%f\n", pow(d1, d2));

}
```

Program Output:



```
Q:\example\c-library-functions.exe
10
0.990105
0.140332
7.389056
-1145744106
81.000000
-
```

Variable scope:

A scope is a region of the program, and the scope of variables refers to the area of the program where the variables can be accessed after its declaration.

In C each and every variable defined in a scope. You can define scope as the section or region of a program where a variable has its existence; moreover, that variable cannot be used or accessed beyond that region.

In C programming, variable declared within a function is different from a variable declared outside of a function. The variable can be declared in three places. These are:

Position	Type
Inside a function or a block.	local variables
Out of all functions.	Global variables
In the function parameters.	Formal parameters

So, now let's have a look at each of them individually.

Local Variables

Variables that are declared within the function block and can be used only within the function is called local variables.

Local Scope or Block Scope

A local scope or block is a collective program statements put in and declared within a function or block (a specific region enclosed with curly braces) and variables lying inside such blocks are termed as local variables. All these locally scoped statements are written and enclosed within left ({) and right braces (}) curly braces. There's a provision for nested blocks also in C which means there can be a block or a function within another block or function. So it can be said that variable(s) that are declared within a block can be accessed within that specific block and all other inner blocks of that block, but those variables cannot be accessed outside the block.

Example:

```
#include <stdio.h>

int main ()

{
```

```
/* local variable definition and initialization */    int x,y,z;

/* actual initialization */    x = 20;

y = 30;

z = x + y;

printf ("value of x = %d, y = %d and z = %d\n", x, y, z);

return 0;

}
```

Global Variables

Variables that are declared outside of a function block and can be accessed inside the function is called global variables.

Global Scope

Global variables are defined outside a function or any specific block, in most of the case, on the top of the C program. These variables hold their values all through the end of the program and are accessible within any of the functions defined in your program.

Any function can access variables defined within the global scope, i.e., its availability stays for the entire program after being declared.

Example:

```
#include <stdio.h>

/* global variable definition */int z;
```

```
int main ()
{
    /* local variable definition and initialization */    int x,y;

    /* actual initialization */    x = 20;

    y = 30;

    z = x + y;

    printf ("value of x = %d, y = %d and z = %d\n", x, y, z);

    return 0;
}
```

Global Variable Initialization

After defining a local variable, the system or the compiler won't be initializing any value to it. You have to initialize it by yourself. It is considered good programming practice to initialize variables before using. Whereas in contrast, global variables get initialized automatically by the compiler as and when defined. Here's how based on datatype; global variables are defined.

datatype	Initial Default Value
int	0
char	'\0'

float	0
double	0
pointer	NULL

Custom Header File:

It helps to manage user-defined methods, global variables, and structures in a separate file, which can be used in different modules.

A process to Create Custom Header File in C

For example, I am calling an external function named swap in my main.c file.

Example:

```
#include<stdio.h>

#include"swap.h"

void main ()
{
    int a=20;
    int b=30;
    swap (&a, &b);
    printf ("a=%d\n", a);
    printf ("b=%d\n",b);
}
```

Swap method is defined in swap.h file, which is used to swap two numbers by using a temporary variable.

Example:

```
void swap (int* a, int* b)
{
    int tmp;

    tmp = *a;

    *a = *b;

    *b = tmp;
}
```

Note:

- header file name must have a .h file extension.
- In this example, I have named swap.h header file.
- Instead of writing <swap.h> use this terminology swap.h for include custom header file.
- Both files swap.h and main.c must be in the same folder.

Recursion:

C is a powerful programming language having capabilities like an iteration of a set of statements 'n' number of times. The same concepts can be done using functions also. In this chapter, you will be learning about recursion concept and how it can be used in the C program.

What is Recursion

Recursion can be defined as the technique of replicating or doing again an activity in a self-similar way calling itself again and again, and the process continues till specific condition reaches. In the world of programming, when your program lets you call that specific function from inside that function, then

this concept of calling the function from itself can be termed as recursion, and the function in which makes this possible is called recursive function.

Here's an example of how recursion works in a program:

Example Syntax:

```
void rec_prog(void) {  
  
    rec_prog(); /* function calls itself */  
  
int main(void) {  
  
    rec_prog();  
  
    return 0;  
  
}
```

C program allows you to do such calling of function within another function, i.e., recursion. But when you implement this recursion concept, you have to be cautious in defining an exit or terminating condition from this recursive function, or else it will continue to an infinite loop, so make sure that the condition is set within your program.

Factorial Program

Example:

```
#include<stdio.h>  
  
#include<conio.h>  
  
int fact(int f) {  
  
    if (f & lt; = 1) {  
  
        printf("Calculated Factorial");  
  
        return 1;  
  
}
```



```
}

return f * fact(f - 1);

}

int main(void) {

    int f = 12;

    clrscr();

    printf("The factorial of %d is %d \n", f, fact(f));

    getch();

    return 0;

}
```

Fibonacci Program

Example:

```
#include<stdio.h>

#include<conio.h>

int fibo(int g) {

    if (g == 0) {

        return 0;

    }

    if (g == 1) {

        return 1;

    }

    return fibo(g - 1) + fibo(g - 2);

}
```

```
}

int main(void) {

    int g;

    clrscr();

    for (g = 0; g < 10; g++) {

        printf("\nNumbers are: %d \t ", fibonacci(g));

    }

    getch();

    return 0;

}
```

Arrays:

The array is a data structure in C programming, which can store a fixed-size sequential collection of elements of the same data type.

For example, if you want to store ten numbers then instead of defining ten variables, it's easy to define an array of 10 lengths.

In the C programming language, an array can be **One-Dimensional**, **Two-Dimensional** and **Multidimensional**.

Define an Array in C

Syntax:

```
type arrayName [ size ];
```

This is called a one-dimensional array. An array type can be any valid C data types, and array size must be an integer constant greater than zero.

Example:

```
double amount[5];
```

Initialize an Array in C

Arrays can be initialized at declaration time:

```
int age[5]={22,25,30,32,35};
```

Initializing each element separately in a loop:

```
int myArray[5];

int n = 0;

// Initializing elements of array seperately
for(n=0;n<sizeof(myArray);n++)
{
    myArray[n] = n;
}
```

A Pictorial Representation of the Array:

	0	1	2	3	4
age	22	25	30	32	35

Accessing Array Elements in C

Example:

```
int myArray[5];

int n = 0;
```

```
// Initializing elements of array separately

for(n=0;n<sizeof(myArray);n++)

{

    myArray[n] = n;

}

int a = myArray[3]; // Assigning 3rd element of array value to integer
'a'.
```

Strings:

In C programming, the one-dimensional array of characters are called strings, which is terminated by a null character '\0'.

In C programming, the one-dimensional array of characters are called strings, which is terminated by a null character '\0'.

Strings Declaration in C

There are two ways to declare a string in C programming:

Example:

Through an array of characters.

```
char name[6];
```

Through pointers.

```
char *name;
```

Strings Initialization in C

Example:

```
char name[6] = {'C', 'l', 'o', 'u', 'd', ' '}
```

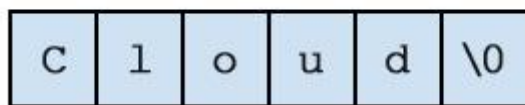
```
char name[6] = {'C', 'l', 'o', 'u', 'd', '\0'};

};
```

or

```
char name[] = "Cloud";
```

Memory Representation of Above Defined String in C



Example:

```
#include<stdio.h>

int main ()
{
    char name[6] = {'C', 'l', 'o', 'u', 'd', ' '
#include<stdio.h>
int main ()
{
    char name[6] = {'C', 'l', 'o', 'u', 'd', '\0'};

    printf("Tutorials%s\n", name );

    return 0;
}

};

printf("Tutorials%s\n", name );

return 0;
}
```

Program Output:

TutorialsCloud

C Pointers:

A pointer is a variable in C, and pointers value is the address of a memory location.

A pointer is a variable in C, and pointers value is the address of a memory location.

Pointer Definition in C

Syntax:

```
type *variable_name;
```

Example:

```
int *width;  
  
char *letter;
```

Benefits of using Pointers in C

- Pointers allow passing of arrays and strings to functions more efficiently.
- Pointers make possible to return more than one value from the function.
- Pointers reduce the length and complexity of a program.
- Pointers increase the processing speed.
- Pointers save the memory.

How to use Pointers in C

Example:

```
#include<stdio.h>  
  
int main ()  
  
{
```

```
int n = 20, *pntr; /* actual and pointer variable declaration */

pntr = &n; /* store address of n in pointer variable*/

printf("Address of n variable: %x\n", &n );

/* address stored in pointer variable */ printf("Address stored in
pntr variable: %x\n", pntr );

/* access the value using the pointer */ printf("Value of *pntr
variable: %d\n", *pntr );

return 0;

}
```

Address of n variable: 2cb60f04

Address stored in pntr variable: 2cb60f04

Value of *pntr variable: 20

Memory Management:

C language provides many functions that come in header files to deal with the allocation and management of memories. In this tutorial, you will find brief information about managing memory in your program using some functions and their respective header files.

C language provides many functions that come in header files to deal with the allocation and management of memories. In this tutorial, you will find brief information about managing memory in your program using some functions and their respective header files.

Management of Memory

Almost all computer languages can handle system memory. All the variables used in your program occupies a precise memory space along with the program itself, which needs some memory for storing itself (i.e., its own

program). Therefore, managing memory utmost care is one of the major tasks a programmer must keep in mind while writing codes.

When a variable gets assigned in a memory in one program, that memory location cannot be used by another variable or another program. So, C language gives us a technique of allocating memory to different variables and programs.

There are two types used for allocating memory. These are:

static memory allocations

In static memory allocation technique, allocation of memory is done at compilation time, and it stays the same throughout the entire run of your program. Neither any changes will be there in the amount of memory nor any change in the location of memory.

dynamic memory allocations

In dynamic memory allocation technique, allocation of memory is done at the time of running the program, and it also has the facility to increase/decrease the memory quantity allocated and can also release or free the memory as and when not required or used. Reallocation of memory can also be done when required. So, it is more advantageous, and memory can be managed efficiently.

`malloc`, `calloc`, or `realloc` are the three functions used to manipulate memory. These commonly used functions are available through the `stdlib` library so you must include this library to use them.

`malloc`, `calloc`, or `realloc` are the three functions used to manipulate memory. These commonly used functions are available through the `stdlib` library so you must include this library to use them.

C - Dynamic memory allocation functions

Function	Syntax
malloc()	malloc (number *sizeof(int));
calloc()	calloc (number, sizeof(int));
realloc()	realloc (pointer_name, number * sizeof(int));
free()	free (pointer_name);

malloc function

- malloc function is used to allocate space in memory during the execution of the program.
- malloc function does not initialize the memory allocated during execution. It carries garbage value.
- malloc function returns null pointer if it couldn't able to allocate requested amount of memory.

Example program for malloc() in C

Example:

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int main()

{

char *mem_alloc;

/* memory allocated dynamically */mem_alloc = malloc( 15 * sizeof(char)

);
```

```
if(mem_alloc== NULL )
{
printf("Couldn't able to allocate requested memory\n");
}
else
{
strcpy( mem_alloc,"w3schools.in");
}

printf("Dynamically allocated memory content : %s\n", mem_alloc );
free(mem_alloc);
}
```

Program Output:

```
Dynamically allocated memory content : w3schools.in
```

calloc function

- calloc () function and malloc () function is similar. But calloc () allocates memory for zero-initializes. However, malloc () does not.

Example program for calloc() in C

Example:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
```

```
int main()
{
char *mem_alloc;
/* memory allocated dynamically */mem_alloc = calloc( 15, sizeof(char) );

if( mem_alloc== NULL )
{
printf("Couldn't able to allocate requested memory\n");
}
else
{
strcpy( mem_alloc,"w3schools.in");
}

printf("Dynamically allocated memory content : %s\n", mem_alloc );
free(mem_alloc);
}
```

Program Output:

```
Dynamically allocated memory content : w3schools.in
```

realloc function

- realloc function modifies the allocated memory size by malloc and calloc functions to new size.

- If enough space doesn't exist in the memory of current block to extend, a new block is allocated for the full size of reallocation, then copies the existing data to the new block and then frees the old block.

free function

- free function frees the allocated memory by malloc (), calloc (), realloc () functions.

Example program for realloc() and free()

Example:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

int main()
{
char *mem_alloc;

/* memory allocated dynamically */mem_alloc = malloc( 20 * sizeof(char)
);

if( mem_alloc == NULL )
{
printf("Couldn't able to allocate requested memory\n");
}

else
{
strcpy( mem_alloc,"w3schools.in");
}
}
```

```
printf("Dynamically allocated memory content : " \ "%s\n", mem_alloc );  
  
mem_alloc=realloc(mem_alloc,100*sizeof(char));  
  
if( mem_alloc == NULL )  
{  
printf("Couldn't able to allocate requested memory\n");  
}  
else  
{  
strcpy( mem_alloc,"space is extended upto 100 characters");  
}  
printf("Resized memory : %s\n", mem_alloc );  
free(mem_alloc);  
}
```

Program Output:

```
Dynamically allocated memory content : w3schools.in
```

Resized memory: space is extended up to 100 characters.

Structures:

The structure is user-defined data type in C, which is used to store a collection of different kinds of data.

- The structure is something similar to an array; the only difference is array is used to store same data types.
- struct keyword is used to declare the structure in C.
- Variables inside the structure are called members of the structure.

Defining a Structure in C

Syntax:

```
struct structureName  
  
{  
  
    //member definitions  
  
};
```

Example:

```
struct Courses  
  
{  
  
    char  WebSite[50];  
  
    char  Subject[50];  
  
    int   Price;  
  
};
```

Accessing Structure Members in C

Example:

```
#include<stdio.h>  
  
#include<string.h>  
  
struct Courses  
  
{  
  
    char  WebSite[50];  
  
    char  Subject[50];  
  
    int   Price;  
  
};
```

```
void main( )
{
    struct Courses C;

    //Initialization

    strcpy( C.WebSite, "w3schools.in");

    strcpy( C.Subject, "The C Programming Language");

    C.Price = 0;

    //Print

    printf( "WebSite : %s\n", C.WebSite);

    printf( "Book Author : %s\n", C.Subject);

    printf( "Book Price : %d\n", C.Price);

}
```

Program Output:

```
WebSite : w3schools.in
```

Unions:

Unions are user-defined data type in C, which is used to store a collection of different kinds of data, just like a structure. However, with unions, you can only store information in one field at any one time.

- Unions are like structures except it used less memory.
- The keyword union is used to declare the structure in C.
- Variables inside the union are called members of the union.

Defining a Union in C

Syntax:

```
union unionName
{
    //member definitions
};
```

Example:

```
union Courses
{
    char  WebSite[50];
    char  Subject[50];
    int   Price;
};
```

Accessing Union Members in C

Example:

```
#include<stdio.h>
#include<string.h>

union Courses
{
    char  WebSite[50];
    char  Subject[50];
    int   Price;
};
```



```
void main( )
{
    union Courses C;

    strcpy( C.WebSite, "w3schools.in");

    printf( "WebSite : %s\n", C.WebSite);

    strcpy( C.Subject, "The C Programming Language");

    printf( "Book Author : %s\n", C.Subject);

    C.Price = 0;

    printf( "Book Price : %d\n", C.Price);

}
```

Program Output:

```
WebSite : w3schools.in
```

Typedef:

C is such a dominant language of its time and now, that even you can name those primary data type of your own and can create your own named data type by blending data type and its qualifier.

C is such a dominant language of its time and now, that even you can name those primary data type of your own and can create your own named data type by blending data type and its qualifier.

The typedef keyword in C

typedef is a C keyword implemented to tell the compiler for assigning an alternative name to C's already exist data types. This keyword, typedef typically employed in association with user-defined data types in cases if the names of datatypes turn out to be a little complicated or intricate for a programmer to get

or to use within programs. The typical format for implementing this typedef keyword is:

Syntax:

```
typedef <existing_names_of_datatype> <alias__userGiven_name>;
```

Here's a sample code snippet as of how typedef command works in C:

Example:

```
typedef signed long slong;
```

slong in the statement as mentioned above is used for a defining a signed qualified long kind of data type. Now the thing is this 'slong', which is an user-defined identifier can be implemented in your program for defining any signed long variable type within your C program. This means:

Example:

```
slong g, d;
```

will allow you to create two variables name 'g' and 'd' which will be of type signed long and this quality of signed long is getting detected from the slong(typedef), which already defined the meaning of slong in your program.

Various Application of typedef

The concept of typedef can be implemented for defining a user-defined data type with a specific name and type. This typedef can also be used with structures of C language. Here how it looks like:

Syntax:

```
typedef struct  
{  
    type first_member;
```

```
type sec_member;

type thrid_member;

} nameOfType;
```

Here nameOfType correspond to the definition of structure allied with it. Now, this nameOfType can be implemented by declaring a variable of this structure type.

```
nameOfType type1, type2;
```

Simple Program of structure in C with the use of typedef:

Example:

```
#include<stdio.h>
#include<string.h>
typedef struct professor
{
    char p_name[50];
    int p_sal;
} prof;
void main(void)
{
    prof pf;
    printf("\n Enter Professor details: \n \n");
    printf("\n Enter Professor name:\t");
    scanf("% s", pf.p_name);
    printf("\n Enter professor salary: \t");
    scanf("% d", &pf.p_sal);
    printf("\n Input done ! ");
```

```
}

```

Using typedef with Pointers

typedef can be implemented for providing a pseudo name to pointer variables as well. In this below-mentioned code snippet, you have to use the typedef, as it is advantageous for declaring pointers.

```
int* a;
```

The binding of pointer (*) is done to the right here. With this kind of statement declaration, you are in fact declaring an as a pointer of type int (integer).

```
typedef int* ptr;
ptr g, h, i;
```

File Handling:

C files I/O functions handle data on a secondary storage device, such as a hard disk.

C can handle files as Stream-oriented data (Text) files, and System oriented data (Binary) files.

Stream-oriented data files	The data is stored in the same manner as it appears on the screen. The I/O operations like buffering, data conversions, etc. take place automatically.
System-oriented data files	System-oriented data files are more closely associated with the OS and data stored in memory without converting into text format.

C File Operations

Five major operations can be performed on file are:

- Creation of a new file.
- Opening an existing file.
- Reading data from a file.
- Writing data in a file.
- Closing a file.

Steps for Processing a File

- Declare a file pointer variable.
- Open a file using fopen() function.
- Process the file using the suitable function.
- Close the file using fclose() function.

To handling files in C, file input/output functions available in the stdio library are:

Function	Uses/Purpose
fopen	Opens a file.
fclose	Closes a file.
getc	Reads a character from a file
putc	Writes a character to a file
getw	Read integer
putw	Write an integer
fprintf	Prints formatted output to a file
fscanf	Reads formatted input from a file

fgets	Read string of characters from a file
fputs	Write string of characters to file
feof	Detects end-of-file marker in a file

Fopen:

C fopen function is used to open an existing file or create a new file.

The basic format of fopen is:

Syntax:

```
FILE *fopen( const char * filePath, const char * mode );
```

Parameters

- **filePath:** The first argument is a pointer to a string containing the name of the file to be opened.
- **mode:** The second argument is an access mode.

C fopen() access mode can be one of the following values:

Mode	Description
r	Opens an existing text file.
w	Opens a text file for writing if the file doesn't exist then a new file is created.
a	Opens a text file for appending(writing at the end of existing file) and create the file if it does not exist.
r+	Opens a text file for reading and writing.

w+	Open for reading and writing and create the file if it does not exist. If the file exists then ma blank.
a+	Open for reading and appending and create the file if it does not exist. The reading will sta the beginning, but writing can only be appended.

Return Value

C fopen function returns NULL in case of a failure and returns a FILE stream pointer on success.

Example:

```
#include<stdio.h>

int main()
{
    FILE *fp;
    fp = fopen("fileName.txt", "w");
    return 0;
}
```

- The above example will create a file called fileName.txt.
- The w means that the file is being opened for writing, and if the file does not exist then the new file will be created.

Fclose:

fclose() function is C library function and it's used to releases the memory stream, opened by fopen()function.

The basic format of fclose is:

Syntax:

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

Return Value

C fclose returns EOF in case of failure and returns 0 on success.

Example:

```
#include<stdio.h>

int main()
{
    FILE *fp;
    fp = fopen("fileName.txt","w");
    fprintf(fp, "%s", "Sample Texts");
    fclose(fp);
    return 0;
}
```

- The above example will create a file called fileName.txt.
- The w means that the file is being opened for writing, and if the file does not exist then the new file will be created.
- The fprintf function writes Sample Texts text to the file.
- The fclose function closes the file and releases the memory stream.

Getc:

getc() function is C library function, and it's used to read a character from a file that has been opened in read mode by fopen() function.

Syntax:

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


Return Value

- `getc()` function returns next requested object from the stream on success.
- Character values are returned as an unsigned char cast to an int or EOF on end of file or error.
- The function `feof()` and `ferror()` to distinguish between end-of-file and error must be used.

Example:

```
#include<stdio.h>

int main()
{
    FILE *fp = fopen("fileName.txt", "r");

    int ch = getc(fp);

    while (ch != EOF)
    {
        /* To display the contents of the file on the screen */
        putchar(ch);

        ch = getc(fp);
    }

    if (feof(fp))
        printf("\n Reached the end of file.");

    else
        printf("\n Something gone wrong.");

    fclose(fp);

    getchar();

    return 0;
}
```

Putc:

putc() function is C library function, and it's used to write a character to the file. This function is used for writing a single character in a stream along with that it moves forward the indicator's position.

putc() function is C library function, and it's used to write a character to the file. This function is used for writing a single character in a stream along with that it moves forward the indicator's position.

Syntax:

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

Example:

```
int main (void)
{
    FILE * fileName;
    char ch;
    fileName = fopen("anything.txt","wt");
    for (ch = 'D' ; ch <= 'S' ; ch++) {
        putc (ch , fileName);
    }
    fclose (fileName);
    return 0;
}
```

Getw:

C getw function is used to read an integer from a file that has been opened in read mode. It is a file handling function, which is used for reading integer values.

Syntax:

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

putw:

C putw function is used to write an integer to the file.

Syntax:

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

Example:

```
int main (void)
{
    FILE *fileName;

    int i=2, j=3, k=4, n;

    fileName = fopen ("anything.c", "w");

    putw(i, fileName);

    putw(j, fileName);

    putw(k, fileName);

    fclose(fileName);

    fileName = fopen ("test.c", "r");

    while (getw(fileName) != EOF)

    {

        n= getw(fileName);

        printf("Value is %d \t: ", n);

    }

    fclose(fp);
}
```

```
return 0;

}
```

Fprintf:

C printf function pass arguments according to the specified format to the file indicated by the stream. This function is implemented in file related programs for writing formatted data in any file.

Syntax:

```
int fprintf(FILE *stream, const char *format, ...)
```

Example:

```
int main (void)
{
    FILE *fileName;

    fileName = fopen("anything.txt","r");

    fprintf(fileName, "%s %s %d", "Welcome", "to", 2018);

    fclose(fileName);

    return(0);
}
```

Fscanf:

C fscanf function reads formatted input from a file. This function is implemented in file related programs for reading formatted data from any file that is specified in the program.

Syntax:

```
int fscanf(FILE *stream, const char *format, ...)
```

Its return the number of variables that are assigned values, or EOF if no assignments could be made.

Example:

```
int main()
{
    char s1[10], s2[10];

    int yr;

    FILE* fileName;

    fileName = fopen("anything.txt", "w+");

    fputs("Welcome to", fileName);

    rewind(fileName);

    fscanf(fileName, "%s %s %d", str1, str2, &yr);

    printf("----- \n");

    printf("1st word %s \t", str1);

    printf("2nd word %s \t", str2);

    printf("Year-Name %d \t", yr);

    fclose(fileName);

    return (0);
}
```

Fgets:

C fgets function is implemented in file related programs for reading strings from any particular file. It gets the strings 1 line each time.

Syntax:

```
char *fgets(char *str, int n, FILE *stream)
```

Example:

```
void main(void)
{
    FILE* fileName;

    char ch[100];

    fileName = fopen("anything.txt", "r");

    printf("%s", fgets(ch, 50, fileName));

    fclose(fileName);
}
```

- On success, the function returns the same str parameter
- C fgets function returns a NULL pointer in case of a failure.

Fputs:

C fputs function is implemented in file related programs for writing string to any particular file.

Syntax:

```
int fputs(const char *str, FILE *stream)
```

Example:

```
void main(void)
{
    FILE* fileName;

    fileName = fopen("anything.txt", "w");

    fputs("Example: ", fileName);

    fclose(fileName);
}
```

- In this function returns non-negative value, otherwise returns EOF on error.

Feof:

C feof function is used to determine if the end of the file (stream), specified has been reached or not. This function keeps on searching the end of file (eof) in your file program.

Syntax:

```
int feof(FILE *stream)
```

Example:

```
while (!feof(fileName)) {  
    printf("%s", str);  
    fgets(st, 50, fileName);  
}  
  
fclose(fileName)
```

- C feof function returns true in case end of file is reached, otherwise it's return false.

Command Line Arguments:

C makes it possible to pass values from the command line at execution time in your program. In this chapter, you will learn about the use of command-line argument in C.

The main() function is the most significant function of C and C++ languages. This main() is typically defined having a return type of integer and having no parameters; something like this:

Example:

```
int main()  
{  
  
    /* body of the main() function */  
  
}
```

C provides programmers to put command-line arguments within the program, which will allow users to add values at the very start of program execution.

What are Command line arguments?

Command line arguments are the arguments specified after the program name in the operating system's command line, and these arguments values are passed to your program at the time of execution from your operating system. For using this concept in your program, you have to understand the complete declaration of how the main function works with this command-line argument to fetch values that earlier took no arguments with it (main() without any argument).

So you can program the main() is such a way that it can essentially accept two arguments where the first argument denotes the number of command line arguments whereas the second argument denotes the full list of every command line arguments. This is how you can code your command line argument within the parenthesis of main():

Example:

```
int main ( int argc, char *argv [ ] )
```

In the above statement, the command line arguments have been handled via main() function, and you have set the arguments where

- argc (ARGument Count) denotes the number of arguments to be passed and
- argv [] (ARGument Vector) denotes to a pointer array that is pointing to every argument that has been passed to your program.

You must make sure that in your command line argument, argv[0] stores the name of your program, similarly argv[1] gets the pointer to the 1st command line argument that has been supplied by the user, and *argv[n]denotes the last argument of the list.

Program for Command Line Argument

Example:

```
#include <stdio.h>

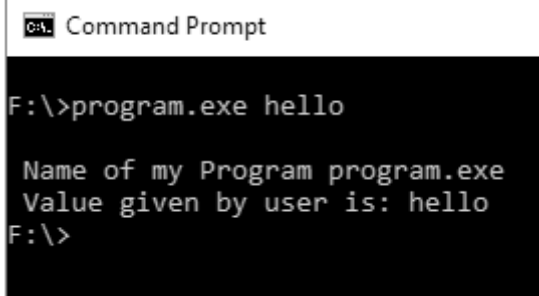
int main( int argc, char *argv [] )
{
    printf(" \n Name of my Program %s \t", argv[0]);

    if( argc == 2 )
    {
        printf("\n Value given by user is: %s \t", argv[1]);
    }

    else if( argc > 2 )
    {
        printf("\n Many values given by users.\n");
    }

    else
    {
        printf(" \n Single value expected.\n");
    }
}
```

Output:



```
C:\> Command Prompt

F:\>program.exe hello

Name of my Program program.exe
Value given by user is: hello
F:\>
```

JBLEETIT