**WHAT IS C ?**

**C language** is a general purpose and structured pragramming langauge developed by 'Dennis Ritchie' at AT &T's Bell Laboratories in the 1972s in USA.

It is also called as 'Procedure oriented programming language.'

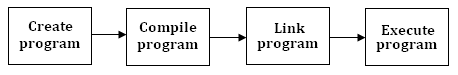
C is not specially designed for specific applications areas like COBOL (Common Business-Oriented Language) or FORTRAN (Formula Translation). It is well suited for business and scietific applications. It has some various features like control structures, looping statements, arrays, macros required for these applications.

The C language has following numorous features as:

* Portability
* Flexibility
* Effectiveness and efficiency
* Reliability
* Interactivity

### EXECUTION OF C PROGRAM :

C program executes in following 4 (four steps).



1. **Creating a program :**

An editor like notepad or wordpad is used to create a C program. This file contains a source code which consists of executable code. The file should be saved as **'\*.c'** extension only.

1. **Compiling the program :**

The next step is to compile the program. The code is compiled by using compiler. Compiler converts executable code to binary code i.e. object code.

1. **Linking a program to library :**

The object code of a program is linked with libraries that are needed for execution of a program. The linker is used to link the program with libraries. It creates a file with **'\*.exe'** extension.

1. **Execution of program :**

The final executable file is then run by dos command prompt or by any other software.

# HISTORY OF C :

|  |  |  |
| --- | --- | --- |
| **Year of Establishment** | **Language Name** | **Developed By** |
| 1960 | ALGOL-60 | Cambridge University |
| 1963 | CPL (Combined Programming Language) | Cambridge University |
| 1967 | BCPL (Basic Combined Programming Language) | Martin Richard at Cambridge University |
| 1970 | B | Ken Thompson at AT & T's Bell Laboratories. |
| 1972 | C | Dennis Ritchie at AT & T' Bell Laboratory. |

The development of C was a cause of evolution of programming languages like Algol 60, CPL (Combined Programming Langauge), BCPL (Basic Combined Programming Language) and B.

* **Algol-60 : (1963)**

ALGOL is an acronym for Algorithmic Language. It was the first structured procedural programming language, developed in the late 1950s and once widely used in Europe. But it was too abstract and too general structured langauage.

* **CPL : (1963)**

CPL is an acronym for Combined Programming Language. It was developed at Cambridge University.

* **BCPL : (1967)**

BCPL is an acronym for Basic Combined Programming Language. It was developed by Martin Richards at Cambridge University in 1967. BCPL was not so powerful. So, it was failed.

* **B : (1970)**

B language was developed by Ken Thompson at AT & T Bell Laboratories in 1970. It was machine dependent. So, it leads to specific problems.

* **C : (1972)**

'C' Programming Langauage was developed by Dennis Ritchie at AT & T Bell Laboratories in 1972. This is general purpose, compiled, structured programming langauage. Dennis Ritchie studied the BCPL, then improved and named it as 'C' which is the second letter of BCPL.

# STRUCTURE OF C PROGRAM

The basic **structure of C program** is as follow:

Document Section

Links Section (File)

Definition Section

Global variable declaration Section

void main()

{  
    Variable declaration section  
    Function declaration section  
    executable statements;  
}

Function definition 1

---------------------

---------------------

Function definition n

where,

**Document Section :** It consists of set of comment lines which include name of a program, author name, creation date and other information.

**Links Section (File) :** It is used to link the required system libraries or header files to excute a program.

**Definition Section :** It is used to define or set values to variables.

**Global variable declaration Section :** It is used to declare global or public variable.

**void main() :** Used to start of actual C program. It includes two parts as declaration part and executable part.

**Variable declaration section :** Used to declare private variable.

**Function declaration section :** Used to declare functions of program from which we get required output.

Then, executable statements are placed for execution.

**Function definition section :** Used to define functions which are to be called from main().

# CHARACTER SET :

A character refers to the digit, alphabet or special symbol used to data represetation.

1. Alphabets :                 A-Z, a-z
2. Digits :                       0-9
3. Special Characters :    ~ ! @ # $ % ^ & \* ( ) \_ + { } [ ] - < > , . / ? \ | : ; " '
4. White Spaces :            Horizontal tab, Carriage return, New line, form feed

### IDENTIFIER :

Identifier is the name of a variable that is made up from combination of alphabets, digits and underscore.

### VARIABLE :

It is a data name which is used to store data and may change during program execution. It is opposite to constant. Variable name is a name given to memory cells location of a computer where data is stored.

**RULES FOR VARIBALES:**

* First character should be letter or alphabet.
* Keywords are not allowed to use as a variable name.
* White space is not allowed.
* C is case sensitive i.e. UPPER and lower case are significant.
* Only underscore, special symbol is allowed between two characters.
* The length of indentifier may be upto 31 characters but only only the first 8 characters are significant by compiler.
* (Note: Some compilers allow variable names whose length may be upto 247 characters. But, it is recommended to use maximum 31 characters in variable name. Large variable name leads to occur errors.)

### KEYWORDS :

* Keywords are the system defined identifiers.
* All keywords have fixed meanings that do not change.
* White spaces are not allowed in keywords.
* Keyword may not be used as an indentifier.
* It is strongly recommended that keywords should be in lower case letters.
* There are totally **32 (Thirty Two) keywords** used in a C programming.

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

### 

### ESCAPE SEQUENCE CHARACTERS (BACKSLASH CHARACTER CONSTANTS) IN C:

C supports some special escape sequence characters that are used to do special tasks.

These are also called as 'Backslash characters'.

Some of the escape sequence characters are as follow:

|  |  |
| --- | --- |
| **Character Constant** | **Meaning** |
| \n | New line (Line break) |
| \b | Backspace |
| \t | Horizontal Tab |
| \f | Form feed |
| \a | Alert (alerts a bell) |
| \r | Carriage Return |
| \v | Vertical Tab |
| \? | Question Mark |
| \' | Single Quote |
| \'' | Double Quote |
| \\ | Backslash |
| \0 | Null |

# CONSTANTS IN C :

A constant is an entity that doesn't change during the execution of a program.

Followings are the different types of constants :

## 1. Real Constant :

* It must have at least one digit.
* It must have a decimal point which may be positive or negative.
* Use of blank space and comma is not allowed between real constants.
* Example:

+194.143, -416.41

## 2. Integer Constant :

* It must have at least one digit.
* It should not contain a decimal place.
* It can be positive or negative.
* Use of blank space and comma is not allowed between real constants.
* Example:

1990, 194, -394

## 3. Character Constant :

* It is a single alphabet or a digit or a special symbol enclosed in a single quote.
* Maximum length of a character constant is 1.
* Example:

'T', '9', '$'

## 4. String Constant :

* It is collection of characters enclosed in double quotes.
* It may contain letters, digits, special characters and blank space.
* Example:

"Technowell Web Solutions, Sangli"

# DATA TYPES IN C :

"Data type can be defined as the type of data of variable or constant store."

When we use a variable in a program then we have to mention the type of data. This can be handled using data type in C.

Followings are the most commonly used data types in C.

|  |  |  |  |
| --- | --- | --- | --- |
| **Keyword** | **Format Specifier** | **Size** | **Data Range** |
| char | %c | 1 Byte | -128 to +127 |
| unsigned char | <-- -- > | 8 Bytes | 0 to 255 |
| int | %d | 2 Bytes | -32768 to +32767 |
| long int | %ld | 4 Bytes | -231 to +231 |
| unsigned int | %u | 2 Bytes | 0 to 65535 |
| float | %f | 4 Bytes | -3.4e38 to +3.4e38 |
| double | %lf | 8 Bytes | -1.7e38 to +1.7e38 |
| long double | %Lf | 12-16 Bytes | -3.4e38 to +3.4e38 |

## QUALIFIER :

When qualifier is applied to the data type then it changes its size or its size.

Size qualifiers : **short, long**

Sign qualifiers : **signed, unsigned**

## ENUM DATA TYPE :

This is an user defined data type having finite set of enumeration constants. The keyword 'enum' is used to create enumerated data type.

Syntax:

enum [data\_type] {const1, const2, ...., const n};

Example:

enum mca(software, web, seo);

## TYPEDEF :

It is used to create new data type. But it is commonly used to change existing data type with another name.

Syntax:

typedef [data\_type] synonym;

OR

typedef [data\_type] new\_data\_type;

Example:

typedef int integer;  
integer rno;

# OPERATORS IN C :

"Operator is a symbol that is used to perform mathematical operations."

When we use a variable in a program then we have to mention the type of data. This can be handled using data type in C.

Followings are the most commonly used data types in C.

|  |  |
| --- | --- |
| **Operator Name** | **Operators** |
| Assignment | = |
| Arithmetic | +, -, \*, /, % |
| Logical | &&, ||, ! |
| Relational | <, >, <=, >=, ==, != |
| Shorthand | +=, -=, \*=, /=, %= |
| Unary | ++, -- |
| Conditional | ()?:; |
| Bitwise | &, |, ^, <<, >>, ~ |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | [1. Assignment Operator](javascript:%20void%200;) | | |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | [2. Arithmetic Operators](javascript:%20void%200;) | | |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | [3. Logical Operators](javascript:%20void%200;) | | |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | [4. Relational Operators](javascript:%20void%200;) | | |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | [5. Shorthand Operators](javascript:%20void%200;) | | |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | [6. Unary Operators](javascript:%20void%200;) | | |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | [7. Conditional Operator](javascript:%20void%200;) | | |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | [8. Bitwise Operators](javascript:%20void%200;) | | |

# OPERATORS PRECEDENCE AND ASSOCIATIVITY :

In C, each and every operator has a spcial precedence which is associated with it. There are various levels of precedence. This precedence is especially used to determine to evaluation of expression which has more than one operator in it. The operators which has higher precedence are executed first and vice-versa. Operators which has same precedence level are evaluated from left to right. It is dependant on it's level. This feature is well known as **'Associativity of an operator.'**

|  |  |  |
| --- | --- | --- |
| **Associativity** | **Operator** | **Description** |
| Left to Right | () | Function |
| [] | Array |
| --> | Pointer to member |
| . | Structure |
|  | | |
| Right to left | - | Unary Minus |
| + | Unary Plus |
| ++ / -- | Increment/Decrement |
| ~ | One's Complement |
| & | Address of |
| (type) | Type casting |
| sizeof | Size (in bytes) |
| ! | Logical Not |
| \* | Pointer reference |
|  | | |
| Left to Right | \* | Multiplication |
| / | Division |
| % | Modulus |
|  | | |
| Left to Right | + | Addition |
| - | Subtraction |
|  | | |
| Left to Right | << | Left Shift |
| >> | Right Shift |
|  | | |
| Left to Right | < | Less than |
| <= | Less than or equal to |
| > | Greater than |
| >= | Greater than or equal to |
|  | | |
| Left to Right | == | Equality |
| != | Not Equal to |
|  | | |
| Left to Right | & | Bitwise AND |
|  | | |
| Left to Right | ^ | Bitwise XOR |
|  | | |
| Left to Right | | | Bitwise OR |
|  | | |
| Left to Right | && | Logical AND |
|  | | |
| Left to Right | || | Logical OR |
|  | | |
| Left to Right | ? : | Conditional Operator |
|  | | |
| Right to Left | =  \*=  += | Assignment |
|  | | |
| Left to Right | , | Comma |

**Precedence and Associativity of operators**

# DECISION MAKING STATEMENTS / CONDITIONAL STATEMENTS :

C program executes program sequentially. Sometimes, a program requires checking of certain conditions in program execution. C provides various key condition statements to check condition and execute statements according conditional criteria.

These statements are called as 'Decision Making Statements' or 'Conditional Statements.'

Followings are the different conditional statements used in C.

1. [If Statement](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/if-statement.asp)
2. [If-Else Statement](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/if-else-statement.asp)
3. [Nested If-Else Statement](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/nested-if-else-statement.asp)
4. [Switch Case](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/switch-case.asp)

# IF STATEMENT :

This is a conditional statement used in C to check condition or to control the flow of execution of statements. This is also called as 'decision making statement or control statement.' The execution of a whole program is done in one direction only.

**Syntax:**

if(condition)

{

statements;

}

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the braces and executes the block of statements associated with it. If it returns false, then program skips the braces. If there are more than 1 (one) statements in if statement then use { } braces else it is not necessary to use.

#### Program :

/\* Program to demonstrate if statement.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int a;

a=5;

clrscr();

if(a>4)

printf("\nValue of A is greater than 4 !");

if(a==4)

printf("\n\n Value of A is 4 !");

getch();

}

#### Output :

Value of A is greater than 4 !\_

# IF-ELSE STATEMENT :

This is also one of the most useful conditional statement used in C to check conditions.

**Syntax:**

if(condition)

{

true statements;

}

else

{

false statements;

}

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the braces and executes the block of statements associated with it. If it returns false, then it executes the else part of a program.

#### Program :

/\* Program to demonstrate if-else statement.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int no;

clrscr();

printf("\n Enter Number :");

scanf("%d",&no);

if(no%2==0)

printf("\n\n Number is even !");

else

printf("\n\n Number is odd !");

getch();

}

#### Output :

Enter Number : 11

Number is odd !\_

# NESTED IF-ELSE STATEMENT :

It is a conditional statement which is used when we want to check more than 1 conditions at a time in a same program. The conditions are executed from top to bottom checking each condition whether it meets the conditional criteria or not. If it found the condition is true then it executes the block of associated statements of true part else it goes to next condition to execute.

**Syntax:**

if(condition)

{

if(condition)

{

statements;

}

else

{

statements;

}

}

else

{

statements;

}

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the braces and again checks the next condition. If it is true then it executes the block of statements associated with it else executes else part.

#### Program :

/\* Program to demonstrate nested if-else statement.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int no;

clrscr();

printf("\n Enter Number :");

scanf("%d",&no);

if(no>0)

{

printf("\n\n Number is greater than 0 !");

}

else

{

if(no==0)

{

printf("\n\n It is 0 !");

}

else

{

printf("Number is less than 0 !");

}

}

getch();

}

#### Output :

Enter Number : 0

It is 0 !\_

# SWITCH CASE STATEMENT :

This is a multiple or multiway brancing decision making statement.

When we use nested if-else statement to check more than 1 conditions then the complexity of a program increases in case of a lot of conditions. Thus, the program is difficult to read and maintain. So to overcome this problem, C provides 'switch case'.

Switch case checks the value of a expression against a case values, if condition matches the case values then the control is transferred to that point.

**Syntax:**

switch(expression)

{

case expr1:

statements;

break;

case expr2:

statements;

break;  
- - - - - - - - - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - - - - - - - - - - - - - -

case exprn:

statements;

break;

default:

statements;

}

In above syntax, switch, case, break are keywords.

expr1, expr2 are known as 'case labels.'

Statements inside case expression need not to be closed in braces.

Break statement causes an exit from switch statement.

Default case is optional case. When neither any match found, it executes.

#### Program :

/\* Program to demonstrate switch case statement.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int no;

clrscr();

printf("\n Enter any number from 1 to 3 :");

scanf("%d",&no);

switch(no)

{

case 1:

printf("\n\n It is 1 !");

break;

case 2:

printf("\n\n It is 2 !");

break;

case 3:

printf("\n\n It is 3 !");

break;

default:

printf("\n\n Invalid number !");

}

getch();

}

#### Output 1:

Enter any number from 1 to 3 : 3

It is 3 !\_

***Output 2 :***

Enter any number from 1 to 3 : 5

Invalid number !\_

## \* Rules for declaring switch case :

* The case label should be integer or character constant.
* Each compound statement of a switch case should contain break statement to exit from case.
* Case labels must end with (:) colon.

## \* Advantages of switch case :

* Easy to use.
* Easy to find out errors.
* Debugging is made easy in switch case.
* Complexity of a program is minimized.

# LOOPING STATEMENTS / ITERATIVE STATEMENTS :

'A loop' is a part of code of a program which is executed repeatedly.

A loop is used using condition. The repetition is done until condition becomes condition true.

A loop declaration and execution can be done in following ways.

* Check condition to start a loop
* Initialize loop with declaring a variable.
* Executing statements inside loop.
* Increment or decrement of value of a variable.

## \* Types of looping statements :

Basically, the types of looping statements depends on the condition checking mode. Condition checking can be made in two ways as : Before loop and after loop. So, there are 2(two) types of looping statements.

* Entry controlled loop
* Exit controlled loop

**1. Entry controlled loop :**

In such type of loop, the test condition is checked first before the loop is executed.

Some common examples of this looping statements are :

* [**while loop**](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/while-loop.asp)
* [**for loop**](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/for-loop.asp)

**2. Exit controlled loop :**

In such type of loop, the loop is executed first. Then condition is checked after block of statements are executed. The loop executed atleat one time compulsarily.

Some common example of this looping statement is :

* [**do-while loop**](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/do-while-loop.asp)

# WHILE LOOP :

This is an entry controlled looping statement. It is used to repeat a block of statements until condition becomes true.

**Syntax:**

while(condition)

{

statements;

increment/decrement;

}

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the loop and executes the block of statements associated with it. At the end of loop increment or decrement is done to change in variable value. This process continues until test condition satisfies.

#### Program :

/\* Program to demonstrate while loop.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int a;

clrscr();

a=1;

while(a<=5)

{

printf("\n GALAXIA");

a+=1 // i.e. a = a + 1

}

getch();

}

#### Output :

GALAXIA

GALAXIA

GALAXIA

GALAXIA

GALAXIA \_

# FOR LOOP :

This is an entry controlled looping statement.

In this loop structure, more than one variable can be initilized. One of the most important feature of this loop is that the three actions can be taken at a time like variable initilisation, condition checking and increment/decrement. The for loop can be more concise and flexible than that of while and do-while loops.

**Syntax:**

for(initialisation; test-condition; incre/decre)

{

statements;

}

In above syntax, the given three expressions are seperated by ';' (Semicolon)

**FEATURES :**

* More concise
* Easy to use
* Highly flexible
* More than one variable can be initilized.
* More than one increments can be applied.
* More than two conditions can be used.

#### Program :

/\* Program to demonstrate for loop.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int a;

clrscr();

for(i=0; i<5; i++)

{

printf("\n\t GALAXIA "); // 5 times

}

getch();

}

#### Output :

GALAXIA

GALAXIA

GALAXIA

GALAXIA

GALAXIA \_

# DO-WHILE LOOP :

This is an exit controlled looping statement.

Sometimes, there is need to execute a block of statements first then to check condition. At that time such type of a loop is used. In this, block of statements are executed first and then condition is checked.

**Syntax:**

do

{

statements;

(increment/decrement);

}while(condition);

In above syntax, the first the block of statements are executed. At the end of loop, while statement is executed. If the resultant condition is true then program control goes to evaluate the body of a loop once again. This process continues till condition becomes true. When it becomes false, then the loop terminates.

**Note: The while statement should be terminated with ; (semicolon).**

#### Program :

/\* Program to demonstrate do while loop.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int a;

clrscr();

a=1;

do

{

printf("\n\t GALAXIA "); // 5 times

a+=1; // i.e. a = a + 1

}while(a<=5);

a=6;

do

{

printf("\n\n\t Technowell"); // 1 time

a+=1; // i.e. a = a + 1

}while(a<=5);

getch();

}

#### Output :

GALAXIA

GALAXIA

GALAXIA

GALAXIA

GALAXIA

Technowell\_

# BREAK STATEMENT :

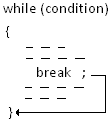
Sometimes, it is necessary to exit immediately from a loop as soon as the condition is satisfied.

When break statement is used inside a loop, then it can cause to terminate from a loop. The statements after break statement are skipped.

**Syntax :**

break;

**Figure :**



#### Program :

/\* Program to demonstrate break statement.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int i;

clrscr();

for(i=1; ; i++)

{

if(i>5)

break;

printf("%d",i); // 5 times only

}

getch();

}

#### Output :

12345\_

# CONTINUE STATEMENT :

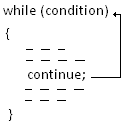
Sometimes, it is required to skip a part of a body of loop under specific conditions. So, C supports 'continue' statement to overcome this anomaly.

The working structure of 'continue' is similar as that of that break statement but difference is that it cannot terminate the loop. It causes the loop to be continued with next iteration after skipping statements in between. Continue statement simply skipps statements and continues next iteration.

**Syntax :**

continue;

**Figure :**



#### Program :

/\* Program to demonstrate continue statement.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int i;

clrscr();

for(i=1; i<=10; i++)

{

if(i==6)

continue;

printf("\n\t %d",i); // 6 is omitted

}

getch();

}

#### Output :

1

2

3

4

5

7

8

9

10\_

# GOTO STATEMENT :

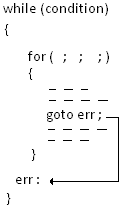
It is a well known as 'jumping statement.' It is primarily used to transfer the control of execution to any place in a program. It is useful to provide branching within a loop.

When the loops are deeply nested at that if an error occurs then it is difficult to get exited from such loops. Simple break statement cannot work here properly. In this situations, goto statement is used.

**Syntax :**

goto [expr];

**Figure :**



#### Program :

/\* Program to demonstrate goto statement.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int i=1, j;

clrscr();

while(i<=3)

{

for(j=1; j<=3; j++)

{

printf(" \* ");

if(j==2)

goto stop;

}

i = i + 1;

}

stop:

printf("\n\n Exited !");

getch();

}

#### Output :

\* \*

Exited\_

### CONTENTS :

1. [Functions](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#func)
2. [Types of Functions :](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#type)
   * [Built In Functions](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#built)
   * [User Defined Functions](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#usr)
3. [Function Call By Passing Value](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#cbpv)
4. [Function Call By Returning Value](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#cbrv)
5. [Function Call By Passing and Returning Value](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#parv)
6. [Advantages](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#adv)
7. [Recursion (Recursive Function)](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp#rec)

# FUNCTIONS IN C :

The function is a self contained block of statements which performs a coherent task of a same kind.

C program does not execute the functions directly. It is required to invoke or call that functions. When a function is called in a program then program control goes to the function body. Then, it executes the statements which are involved in a function body. Therefore, it is possible to call fuction whenever we want to process that functions statements.

### TYPES OF FUNCTIONS :

There are 2(two) types of functions as:

**1. Built in Functions  
2. User Defined Functions**  
  
**1. BUILT IN FUNCTIONS :**

These functions are also called as 'library functions'. These functions are provided by system. These functions are stored in library files. e.g.

* scanf()
* printf()
* strcpy
* strlwr
* strcmp
* strlen
* strcat

**1. USER DEFINED FUNCTIONS :**

The functions which are created by user for program are known as 'User defined functions'.

**Syntax:**

void main()

{

// Function prototype

<return\_type><function\_name>([<argu\_list>]);

// Function Call

<function\_name>([<arguments>]);

}

// Function definition

<return\_type><function\_name>([<argu\_list>]);

{

<function\_body>;

}

#### Program :

/\* Program to demonstrate function.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void add()

{

int a, b, c;

clrscr();

printf("\n Enter Any 2 Numbers : ");

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

c = a + b;

printf("\n Addition is : %d",c);

}

void main()

{

void add();

add();

getch();

}

#### Output :

Enter Any 2 Numbers : 23 6

Addition is : 29\_

### ADVANTAGES :

* It is easy to use.
* Debugging is more suitable for programs.
* It reduces the size of a program.
* It is easy to understand the actual logic of a program.
* Highly suited in case of large programs.
* By using functions in a program, it is possible to construct modular and structured programs.

# FUNCTION CALL BY PASSING VALUE :

When a function is called by passing value of variables then that function is known as 'function call by passing values.'

**Syntax:**

// Declaration

void <function\_name>(<data\_type><var\_nm>);

// Calls

<function\_name>(<var\_nm>);

// Definition

void <function\_name>(<data\_type><var\_nm>);

{

<function\_body>;

- - - - - - - -;

}

#### Program :

/\* Program to demonstrate function call by passing value.

Creation Date :

Author : \*/

#include <stdio.h>

#include <conio.h>

void printno(int a)

{

printf("\n Number is : %d", a);

}

void main()

{

int no;

void printno(int);

clrscr();

printf("\n Enter Number : ");

scanf("%d", &no);

printno(no);

getch();

}

#### Output :

Enter Number : 21

Number is : 21\_

# FUNCTION CALL BY RETURNING VALUE :

When a function returns value of variables then that function is known as 'function call by returning values.'

**Syntax:**

// Declaration

<data\_type><function\_name>();

// Calls

<variable\_of\_function>=<function\_nm>();

// Definition

<data\_type><function\_name>()

{

<function\_body>;

- - - - - - - -;

return <variable\_of\_function>;

}

#### Program :

/\* Program to demonstrate function call by returning value.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

int number()

{

int no;

printf("\n Enter Number : ");

scanf("%d",&no);

return no;

}

void main()

{

int no;

int number();

clrscr();

no = number();

printf("\n Number is : %d",no);

getch();

}

#### Output :

Enter Number : 5

Number is : 5\_

# FUNCTION CALL BY PASSING AND RETURNING VALUE :

When a function passes and returns value of variables then that function is known as 'function call by passing and returning values.'

#### Program :

/\* Program to demonstrate function call by passing and returning value.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

int number(int n)

{

return n;

}

void main()

{

int number(int);

int a = number(4);

clrscr();

printf("\n Number is : %d",a);

getch();

}

#### Output :

Number is : 4\_

# RECURSION (RECURSIVE FUNCTION) :

When a function of body calls the same function then it is called as 'recursive function.'

**Example:**

Recursion()

{

printf("Recursion !");

Recursion();

}

#### Program :

/\* Program to demonstrate function recursion.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

Recursion()

{

int no;

printf("\nRecursion... ");

printf("\n\n Enter Number : ");

scanf("%d",&no);

if (no==3)

exit(0);

else

Recursion();

}

void main()

{

clrscr();

Recursion();

}

#### Output :

Recursion...

Enter Number : 2

Recursion...

Enter Number : 1

Recursion...

Enter Number : 3\_

### Features :

* There should be at least one if statement used to terminate recursion.
* It does not contain any looping statements.

### Advantages :

* It is easy to use.
* It represents compact programming strctures.

### Disadvantages :

* It is slower than that of looping statements because each time function is called.

### Note :

* It can be applied to calculate factorial of a number, fibonancci series.

# STORAGE CLASS :

'Storage' refers to the scope of a variable and memory allocated by compiler to store that variable. Scope of a variable is the boundary within which a varible can be used. Storage class defines the the scope and lifetime of a variable.

From the point view of C compiler, a variable name identifies physical location from a computer where varaible is stored. There are two memory locations in a computer system where variables are stored as : Memory and CPU Registers.

**FUNCTIONS OF STORAGE CLASS :**

To detemine the location of a variable where it is stored ?

Set initial value of a variable or if not specified then setting it to default value.

Defining scope of a variable.

To determine the life of a variable.

### TYPES OF STORAGE CLASSES :

Storage classes are categorised in 4 (four) types as,

* [Automatic Storage Class](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/automatic-storage-class.asp)
* [Register Storage Class](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/register-storage-class.asp)
* [Static Storage Class](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/static-storage-class.asp)
* [External Storage Class](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/external-storage-class.asp)

# AUTOMATIC STORAGE CLASS :

* Keyword : auto
* Storage Location : Main memory
* Initial Value : Garbage Value
* Life : Control remains in a block where it is defined.
* Scope : Local to the block in which variable is declared.

**Syntax :**

auto [data\_type] [variable\_name];

**Example :**

auto int a;

#### Program :

/\* Program to demonstrate automatic storage class.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

auto int i=10;

clrscr();

{

auto int i=20;

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

}

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

getch();

}

#### Output :

20

10\_

# REGISTER STORAGE CLASS :

* Keyword : register
* Storage Location : CPU Register
* Initial Value : Garbage
* Life : Local to the block in which variable is declared.
* Scope : Local to the block.

**Syntax :**

register [data\_type] [variable\_name];

**Example :**

register int a;

When the calculations are done in CPU, then the value of variables are transferred from main memory to CPU. Calculations are done and the final result is sent back to main memory. This leads to slowing down of processes.

Register variables occur in CPU and value of that register variable is stored in a register within that CPU. Thus, it increases the resultant speed of operations. There is no waste of time, getting variables from memory and sending it to back again.

It is not applicable for arrays, structures or pointers.

It cannot not used with static or external storage class.

Unary and address of (&) cannot be used with these variables as explicitly or implicitly.

#### Program :

/\* Program to demonstrate register storage class.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

register int i=10;

clrscr();

{

register int i=20;

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

}

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

getch();

}

#### Output :

20

10\_

# STATIC STORAGE CLASS :

* Keyword : static
* Storage Location : Main memory
* Initial Value : Zero and can be initialize once only.
* Life : depends on function calls and the whole application or program.
* Scope : Local to the block.

**Syntax :**

static [data\_type] [variable\_name];

**Example :**

static int a;

There are two types of static variables as :   
a) Local Static Variable  
b) Global Static Variable

Static storage class can be used only if we want the value of a variable to persist between different function calls.

#### Program :

/\* Program to demonstrate static storage class.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int i;

void incre(void);

clrscr();

for (i=0; i<3; i++)

incre();

getch();

}

void incre(void)

{

int avar=1;

static int svar=1;

avar++;

svar++;

printf("\n\n Automatic variable value : %d",avar);

printf("\t Static variable value : %d",svar);

}

#### Output :

Automatic variable value : 2 Static variable value : 2

Automatic variable value : 2 Static variable value : 3

Automatic variable value : 2 Static variable value : 4\_

# EXTERNAL STORAGE CLASS :

* Keyword : extern
* Storage Location : Main memory
* Initial Value : Zero
* Life : Until the program ends.
* Scope : Global to the program.

**Syntax :**

extern [data\_type] [variable\_name];

**Example :**

extern int a;

The variable access time is very fast as compared to other storage classes. But few registers are available for user programs.

The variables of this class can be referred to as 'global or external variables.' They are declared outside the functions and can be invoked at anywhere in a program.

#### Program :

/\* Program to demonstrate external storage class.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

extern int i=10;

void main()

{

int i=20;

void show(void);

clrscr();

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

show();

getch();

}

void show(void)

{

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

}

#### Output :

20

10\_

# ARRAY :

Array is a collection of homogenous data stored under unique name. The values in an array is called as 'elements of an array.' These elements are accessed by numbers called as 'subscripts or index numbers.' Arrays may be of any variable type.

Array is also called as 'subscripted variable.'

### TYPES OF AN ARRAY :

1. [One / Single Dimensional Array](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/one-single-dimensional-array-in-c.asp#oned)
2. [Two Dimensional Array](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/two-dimensional-array.asp)

# SINGLE / ONE DIMENSIONAL ARRAY :

The array which is used to represent and store data in a linear form is called as 'single or one dimensional array.'

**Syntax:**

<data-type> <array\_name> [size];

**Example:**

int a[3] = {2, 3, 5};

char ch[20] = "TechnoExam" ;

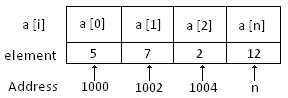
float stax[3] = {5003.23, 1940.32, 123.20} ;

**Total Size (in Bytes):**

total size = length of array \* size of data type

In above example, a is an array of type integer which has storage size of 3 elements. The total size would be 3 \* 2 = 6 bytes.

## Memory Allocation :



Memory allocation for one dimensional array

#### Program :

/\* Program to demonstrate one dimensional array.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int a[3], i;;

clrscr();

printf("\n\t Enter three numbers : ");

for(i=0; i<3; i++)

{

scanf("%d", &a[i]); // read array

}

printf("\n\n\t Numbers are : ");

for(i=0; i<3; i++)

{

printf("\t %d", a[i]); // print array

}

getch();

}

#### Output :

Enter three numbers : 9 4 6

Numbers are : 9 4 6\_

**Features :**

* Array size should be positive number only.
* String array always terminates with null character ('\0').
* Array elements are countered from 0 to n-1.
* Useful for multiple reading of elements (numbers).

**Disadvantages :**

* There is no easy method to initialize large number of array elements.
* It is difficult to initialize selected elements.

# TWO DIMENSIONAL ARRAY :

The array which is used to represent and store data in a tabular form is called as 'two dimensional array.' Such type of array specially used to represent data in a matrix form.

The following syntax is used to represent two dimensional array.

**Syntax:**

<data-type> <array\_nm> [row\_subscript][column-subscript];

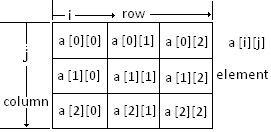
**Example:**

int a[3][3];

In above example, a is an array of type integer which has storage size of 3 \* 3 matrix. The total size would be 3 \* 3 \* 2 = 18 bytes.

It is also called as 'multidimensional array.'

## Memory Allocation :



Memory allocation for two dimensional array

#### Program :

/\* Program to demonstrate two dimensional array.  
Creation Date :

Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int a[3][3], i, j;

clrscr();

printf("\n\t Enter matrix of 3\*3 : ");

for(i=0; i<3; i++)

{

for(j=0; j<3; j++)

{

scanf("%d",&a[i][j]); //read 3\*3 array

}

}

printf("\n\t Matrix is : \n");

for(i=0; i<3; i++)

{

for(j=0; j<3; j++)

{

printf("\t %d",a[i][j]); //print 3\*3 array

}

printf("\n");

}

getch();

}

#### Output :

Enter matrix of 3\*3 : 3 4 5 6 7 2 1 2 3

Matrix is :

3 4 5

6 7 2

1 2 3\_

**Limitations of two dimensional array :**

* We cannot delete any element from an array.
* If we dont know that how many elements have to be stored in a memory in advance, then there will be memory wastage if large array size is specified.

# STRUCTURE :

**Structure** is user defined data type which is used to store heterogeneous data under unique name. Keyword 'struct' is used to declare structure.

The variables which are declared inside the structure are called as 'members of structure'.

**Syntax:**

struct structure\_nm

{

<data-type> element 1;

<data-type> element 2;

- - - - - - - - - - -

- - - - - - - - - - -

<data-type> element n;

}struct\_var;

**Example :**

struct emp\_info

{

char emp\_id[10];

char nm[100];

float sal;

}emp;

**Note :**

1. Structure is always terminated with semicolon (;).

2. Structure name as emp\_info can be later used to declare structure variables of its type in a program.

## Instances of Structure :

Instances of structure can be created in two ways as,

**Instance 1:**

struct emp\_info

{

char emp\_id[10];

char nm[100];

float sal;

}emp;

**Instance 2:**

struct emp\_info

{

char emp\_id[10];

char nm[100];

float sal;

};

struct emp\_info emp;

In above example, emp\_info is a simple structure which consists of stucture members as Employee ID(emp\_id), Employee Name(nm), Employee Salary(sal).

## Aceessing Structure Members :

Structure members can be accessed using member operator '**.**' . It is also called as '**dot operator**' or '**period operator**'.

structure\_var.member;

#### Program :

/\* Program to demonstrate structure.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

struct comp\_info

{

char nm[100];

char addr[100];

}info;

void main()

{

clrscr();

printf("\n Enter Company Name : ");

gets(info.nm);

printf("\n Enter Address : ");

gets(info.addr);

printf("\n\n Company Name : %s",info.nm);

printf("\n\n Address : %s",info.addr);

getch();

}

#### Output :

Enter Company Name : GALAXIA INFOTECH

Enter Address : NEAR BOYS POLYTECHNIC , RANCHI

Company Name : GALAXIA INFOTECH

Address : NEAR BOYS POLYTECHNIC , RANCHI

# Array in Structures :

Sometimes, it is necessary to use structure members with array.

#### Program :

/\* Program to demonstrate array in structures.  
Creation Date :

Author : \*/

#include <stdio.h>

#include <conio.h>

struct result

{

int rno, mrks[5];

char nm;

}res;

void main()

{

int i,total;

clrscr();

total = 0;

printf("\n\t Enter Roll Number : ");

scanf("%d",&res.rno);

printf("\n\t Enter Marks of 3 Subjects : ");

for(i=0;i<3;i++)

{

scanf("%d",&res.mrks[i]);

total = total + res.mrks[i];

}

printf("\n\n\t Roll Number : %d",res.rno);

printf("\n\n\t Marks are :");

for(i=0;i<3;i++)

{

printf(" %d",res.mrks[i]);

}

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

getch();

}

#### Output :

Enter Roll Number : 1

Enter Marks of 3 Subjects : 63 66 68

Roll Number : 1

Marks are : 63 66 68

Total is : 197\_

# Structure With Array :

We can create structures with array for ease of operations in case of getting multiple same fields.

#### Program :

/\* Program to demonstrate Structure With Array.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

struct emp\_info

{

int emp\_id;

char nm[50];

}emp[2];

void main()

{

int i;

clrscr();

for(i=0;i<2;i++)

{

printf("\n\n\t Enter Employee ID : ");

scanf("%d",&emp[i].emp\_id);

printf("\n\n\t Employee Name : ");

scanf("%s",emp[i].nm);

}

for(i=0;i<2;i++)

{

printf("\n\t Employee ID : %d",emp[i].emp\_id);

printf("\n\t Employee Name : %s",emp[i].nm);

}

getch();

}

#### Output :

Enter Employee ID : 1

Employee Name : ABC

Enter Employee ID : 2

Employee Name : XYZ

Employee ID : 1

Employee Name : ABC

Employee ID : 2

Employee Name : XYZ\_

# Structures within Structures (Nested Structures) :

Structures can be used as structures within structures. It is also called as 'nesting of structures'.

**Syntax:**

struct structure\_nm

{

<data-type> element 1;

<data-type> element 2;

- - - - - - - - - - -

- - - - - - - - - - -

<data-type> element n;

struct structure\_nm

{

<data-type> element 1;

<data-type> element 2;

- - - - - - - - - - -

- - - - - - - - - - -

<data-type> element n;

}inner\_struct\_var;

}outer\_struct\_var;

**Example :**

struct stud\_Res

{

int rno;

char nm[50];

char std[10];

struct stud\_subj

{

char subjnm[30];

int marks;

}subj;

}result;

In above example, the structure stud\_Res consists of stud\_subj which itself is a structure with two members. Structure stud\_Res is called as 'outer structure' while stud\_subj is called as 'inner structure.' The members which are inside the inner structure can be accessed as follow :

result.subj.subjnm

result.subj.marks

#### Program :

/\* Program to demonstrate nested structures.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

struct stud\_Res

{

int rno;

char std[10];

struct stud\_Marks

{

char subj\_nm[30];

int subj\_mark;

}marks;

}result;

void main()

{

clrscr();

printf("\n\t Enter Roll Number : ");

scanf("%d",&result.rno);

printf("\n\t Enter Standard : ");

scanf("%s",result.std);

printf("\n\t Enter Subject Code : ");

scanf("%s",result.marks.subj\_nm);

printf("\n\t Enter Marks : ");

scanf("%d",&result.marks.subj\_mark);

printf("\n\n\t Roll Number : %d",result.rno);

printf("\n\n\t Standard : %s",result.std);

printf("\nSubject Code : %s",result.marks.subj\_nm);

printf("\n\n\t Marks : %d",result.marks.subj\_mark);

getch();

}

#### Output :

Enter Roll Number : 1

Enter Standard : MCA(Sci)-I

Enter Subject Code : SUB001

Enter Marks : 63

Roll Number : 1

Standard : MCA(Sci)-I

Subject Code : SUB001

Marks : 63\_

# POINTER :

Pointer is a variable which holds the memory address of another variable. Pointers are represented by '\*'. It is a derive data type in C. Pointer returns the value of stored address.

**Syntax:**

<data\_type> \*pointer\_name;

In above syntax,   
\* = variable pointer\_name is a pointer variable.   
pointer\_name requires memory location   
pointer\_name points to a variable of type data type.

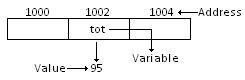
**HOW TO USE ?**

int \*tot;

**Illustration :**

int tot = 95;

**Figure :**



In above example, the statement instructs the system to find out a location for integer variable quantity and puts the values 95 in that memory location.

**Features of Pointer :**  
\* Pointer variable should have prefix '\*'.  
\* Combination of data types is not allowed.  
\* Pointers are more effective and useful in handling arrays.  
\* It can also be used to return multiple values from a funtion using function arguments.  
\* It supports dynamic memory management.  
\* It reduces complexity and length of a program.  
\* It helps to improve execution speed that results in reducing program execution time.

#### Program :

/\* Program to demonstrate pointer.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

void main()

{

int a=10;

int \*ptr;

clrscr();

ptr = &a;

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

scanf("\n\n\t Value of pointer ptr : %d", \*ptr);

printf("\n\n\t Address of pointer ptr : %d", ptr);

getch();

}

#### Output :

Value of a : 10

Value of pointer ptr : 10

Address of pointer ptr : -12\_

# UNION :

Union is user defined data type used to stored data under unique variable name at single memory location.

Union is similar to that of stucture. Syntax of union is similar to stucture. But the major **difference between structure and union is 'storage.'** In structures, each member has its own storage location, whereas all the members of union use the same location. Union contains many members of different types, it can handle only one member at a time.

To declare union data type, 'union' keyword is used.

Union holds value for one data type which requires larger storage among their members.

**Syntax:**

union union\_name

{

<data-type> element 1;

<data-type> element 2;

<data-type> element 3;

}union\_variable;

**Example:**

union student

{

int roll;

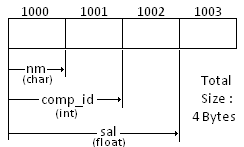
char name[30];

float percentage;

}stu;

In above example, it declares stu variable of type union. The union contains three members as data type of int, char, float. We can use only one of them at a time.

## Memory Allocation :



Memory allocation for union

To access union members, we can use the following syntax.

stu.roll

stu.name

stu.percentage

#### Program :

/\* Program to demonstrate union.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

union galaxia\_infotech

{

int id;

char nm[50];

}emp;

void main()

{

clrscr();

printf("\n\t Enter developer id : ");

scanf("%d", & emp.id);

printf("\n\n\t Enter developer name : ");

scanf("%s", emp.nm);

printf("\n\n Developer ID : %d", emp.id);//Garbage

printf("\n\n Developed By : %s", emp.nm);

getch();

}

#### Output :

Enter developer id : 7

Enter developer name : NANDAN JAISWAL

Developer ID : 7

Developed By : NANDAN JAISWAL

# STRING HANDLING IN C :

**STRING :**

A string is a collection of characters. Strings are always enlosed in double quotes as "string\_constant".

Strings are used in string handling operations such as,

* Counting the length of a string.
* Comparing two strings.
* Copying one string to another.
* Converting lower case string to upper case.
* Converting upper case string to lower case.
* Joining two strings.
* Reversing string.

### Declaration :

The string can be declared as follow :

**Syntax:**

char string\_nm[size];

**Example:**

char name[50];

### String Structure :

When compiler assigns string to character array then it automatically supplies **null character ('\0')** at the end of string. Thus, size of string = original length of string + 1.

char name[8];

name = "GALAXIA";

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ‘G’ | ‘A’ | ‘L’ | ‘A’ | ‘X’ | ‘I’ | ‘A’ | ‘\0’ |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

### Read Strings :

To read a string, we can use scanf() function with format specifier %s.

char name[50];

scanf("%s",name);

The above format allows to accept only string which does not have any blank space, tab, new line, form feed, carriage return.

### Write Strings :

To write a string, we can use printf() function with format specifier %s.

char name[50];

scanf("%s",name);

printf("%s",name);

# string.h header file :

'string.h' is a header file which includes the declarations, functions, constants of string handling utilities. These string functions are widely used today by many programmers to deal with string operations.

Some of the standard member functions of string.h header files are,

#### Program :

/\* Program to demonstrate string.h header file working.  
Creation Date :   
Author : \*/

#include <stdio.h>

#include <conio.h>

#include <string.h>

void main()

{

char str[50];

clrscr();

printf("\n\t Enter your name : ");

gets(str);

printf("\nLower case of string: %s",strlwr(str));

printf("\nUpper case of string: %s",strupr(str));

printf("\nReverse of string: %s",strrev(str));

printf("\nLength of String: %d",strlen(str));

getch();

}

#### Output :

Enter your name : Galaxia

Lower case of string: galaxia

Upper case of string: GALAXIA

Reverse of string: AIXALAG

Length of String: 7\_

# Header File in C :

Header file contains different predefined functions, which are required to run the program. All header files should be included explicitly before main ( ) function.

It allows programmers to seperate functions of a program into reusable code or file. It contains declarations of variables, subroutines. If we want to declare identifiers in more than one source code file then we can declare such identifiers in header file. Header file has extension like '\*.h'. The prototypes of library functions are gathered together into various categories and stored in header files.

E.g. All prototypes of standard input/output functions are stored in header file 'stdio.h' while console input/output functions are stored in 'conio.h'.

The header files can be defined or declared in two ways as

Method 1 : #include "header\_file-name"  
Method 2 : #include <header\_file-name>

Method 1 is used to link header files in current directory as well as specified directories using specific path. The path must be upto 127 characters. This is limit of path declaration. Method 2 is used to link header files in specified directories only.

# Standard Header Files :

Followings are the some commonly used header files which plays a vital role in C programming :

|  |  |  |
| --- | --- | --- |
| [Assert.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/assert.h-file.asp) | [Ctype.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/ctype.h-file.asp) | [Math.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/math.h-file.asp) |
| [Process.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/process.h-file.asp) | [Stdio.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/stdio.h-file.asp) | [Stdlib.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/stdlib.h-file.asp) |
| [String.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/string.h-file.asp) | [Time.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/time.h-file.asp) | [Graphics.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/graphics.h-file.asp) |

**CONTENTS :**

This section contains following key points that discusses the various key terminologies involved in C programming.

* [Contents](http://www.technoexam.com/)
* What is C ? :
  + [What is C ?](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/introduction-to-c-programming.asp)
  + [Execution of C Program](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/introduction-to-c-programming.asp#exe)
* [History](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/history-development-evolution.asp)
* [Structure of C Program](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/basic-structure-of-c-program.asp)
* Variables and Keywords :
  + [Character Set](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/variables-keywords-identifier.asp#techno1)
  + [Identifier](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/variables-keywords-identifier.asp#techno2)
  + [Variable](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/variables-keywords-identifier.asp#techno3)
  + [Keywords](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/variables-keywords-identifier.asp#techno4)
  + [Escape Sequence Characters (Backslash Character Constants) in C](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/variables-keywords-identifier.asp#techno5)
* Constants :
  + [Real Constant](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/constants-in-c.asp#real)
  + [Integer Constant](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/constants-in-c.asp#int)
  + [Character Constant](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/constants-in-c.asp#char)
  + [String Constant](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/constants-in-c.asp#str)
* Data Types :
  + [Data Types](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/data-types-enum-typedef.asp#dtypes)
  + [Qualifier](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/data-types-enum-typedef.asp#qualifier)
  + [Enum](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/data-types-enum-typedef.asp#enum)
  + [Typedef](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/data-types-enum-typedef.asp#typedef)
* Operators :
  + [Assignment Operator](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-conditional-in-c.asp)
  + [Arithmetic Operators](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-conditional-in-c.asp)
  + [Logical Operators](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-conditional-in-c.asp)
  + [Relational Operators](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-conditional-in-c.asp)
  + [Shorthand Operators](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-conditional-in-c.asp)
  + [Unary Operators](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-conditional-in-c.asp)
  + [Conditional / Ternary Operator](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-conditional-in-c.asp)
  + [Biwise Operators](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-conditional-in-c.asp)
  + [Operator Precedence and Associativity](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/operators-precedence-associativity.asp)
* Decision / Control Statements :
  + [Decision / Control Statements](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/conditional-statements-c.asp)
  + [If Statement](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/if-statement.asp)
  + [If-Else Statement](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/if-else-statement.asp)
  + [Nested If-Else](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/nested-if-else-statement.asp)
  + [Switch Case](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/switch-case.asp)
* Looping Statements
  + [Looping / Iterative Statements](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/looping-statements-c.asp)
  + [while](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/while-loop.asp)
  + [do while](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/do-while-loop.asp)
  + [for loop](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/for-loop.asp)
* [Break Statement](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/break-statement.asp)
* [Continue Statement](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/continue-statement.asp)
* [Goto](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/goto-statement.asp)
* Functions :
  + [Functions](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/functions-in-c.asp)
  + [Function call by passing value](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/Function-Call-By-Passing-Value.asp)
  + [Function call by returning value](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/Function-Call-By-Returning-Value.asp)
  + [Function call by passing and returning value](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/Function-Call-By-Passing-and-Returning-Value.asp)
  + [Recursion](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/recursive-function-recursion.asp)
* Storage Classes :
  + [Storage Classes](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/storage-class-of-variable.asp)
  + [Automatic Storage Class (auto)](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/automatic-storage-class.asp)
  + [Register Storage Class (register)](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/register-storage-class.asp)
  + [Static Storage Class (static)](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/static-storage-class.asp)
  + [External Storage Class (extern)](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/external-storage-class.asp)
* Array :
  + [Array](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/one-single-dimensional-array-in-c.asp)
  + [Single / One Dimensional Array](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/one-single-dimensional-array-in-c.asp)
  + [Two Dimensional Array](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/two-dimensional-array.asp)
* Structure :
  + [Structure](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/structure-in-c.asp#struct)
  + [Array in Structures](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/array-in-structures.asp#arrinstrc)
  + [Structure with Array](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/structure-with-array.asp#structarr)
  + [Nested Structure](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/nested-structures-structure-within-structure.asp#neststrct)
  + [Difference between array and structure](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/difference-between-array-and-structure.asp)
* [Pointer](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/pointer-c-programming.asp)
* Union :
  + [Union](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/union-in-c.asp)
  + [Difference between structure and union](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/difference-between-structure-and-union.asp)
* [Strings](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/string-handling-operations.asp)
* [File Handling](http://www.technoexam.com/)
* Header Files :
  + [Header File](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/header-files.asp)
  + [Assert.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/assert.h-file.asp)
  + [Ctype.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/ctype.h-file.asp)
  + [Math.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/math.h-file.asp)
  + [Process.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/process.h-file.asp)
  + [Stdio.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/stdio.h-file.asp)
  + [Stdlib.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/stdlib.h-file.asp)
  + [String.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/string.h-file.asp)
  + [Time.h](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/time.h-file.asp)
* [Programs Library :](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/c-programs-library.asp)
* [Projects Library :](http://www.technoexam.com/c-language-lecture-study-notes-tutorials-material/c-college-projects-library.asp)