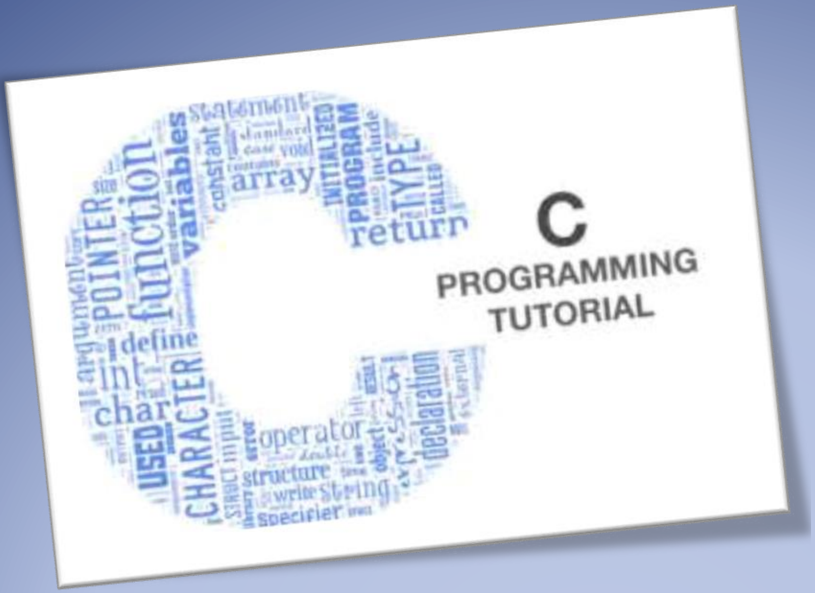


C programming Notes

Enosis Learning

Index

Index

1. Chapter - Introduction Of C

What is C Programming Langauge?

C is a general-purpose programming language that is extremely popular, simple, and flexible to use. It is a structured programming language that is machine-independent and extensively used to write various applications, Operating Systems like Windows, and many other complex programs like Oracle database, Git, Python interpreter, and more.

History of C language

The base or father of programming languages is 'ALGOL.' It was first introduced in 1960. 'ALGOL' was used on a large basis in European countries. 'ALGOL' introduced the concept of structured programming to the developer community.

In 1972, a great computer scientist Dennis Ritchie created a new programming language called 'C' at the Bell Laboratories. It was created from 'ALGOL', 'BCPL' and 'B' programming languages. 'C' programming language contains all the features of these languages and many more additional concepts that make it unique from other languages.

Dennis Ritchie

Where we use C Language:

C Language is mainly used for;

- Design Operating system
- Design Language Compiler
- ‘C’ language is widely used in embedded systems.
- It is used for developing system applications.
- It is widely used for developing desktop applications.
- Most of the applications by Adobe are developed using ‘C’ programming language.
- It is used for developing browsers and their extensions. Google’s Chromium is built using ‘C’ programming language.
- It is used to develop databases. MySQL is the most popular database software which is built using ‘C’.
- It is used in developing an operating system. Operating systems such as Apple’s OS X, Microsoft’s Windows, and Symbian are developed using ‘C’ language. It is used for developing desktop as well as mobile phone’s operating system.
- It is used for compiler production.

Design Database

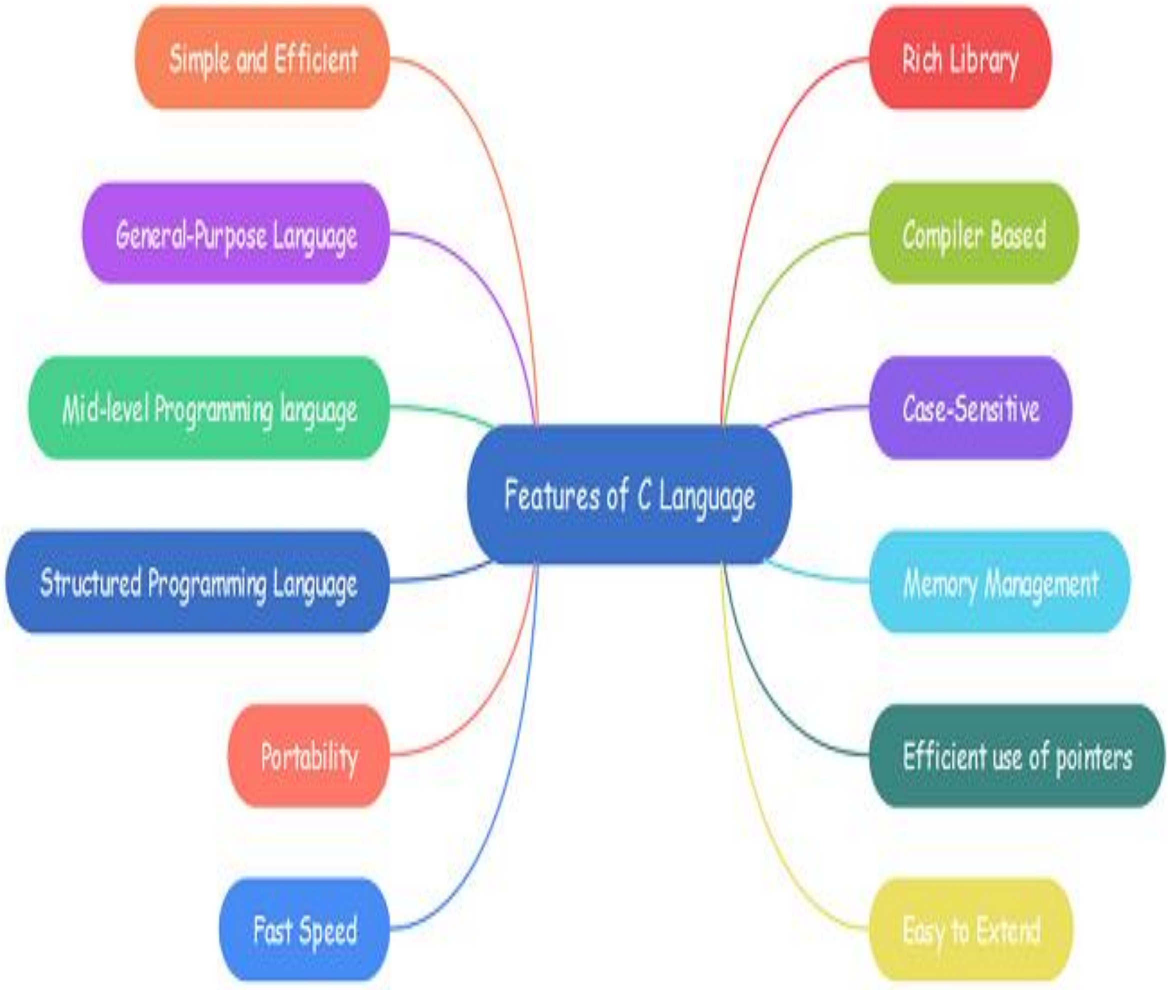
- Language Interpreters
- Utilities
- Network Drivers
- Assemblers

Applications of C:

- C programming language can be used to design the system software like operating system and Compiler.

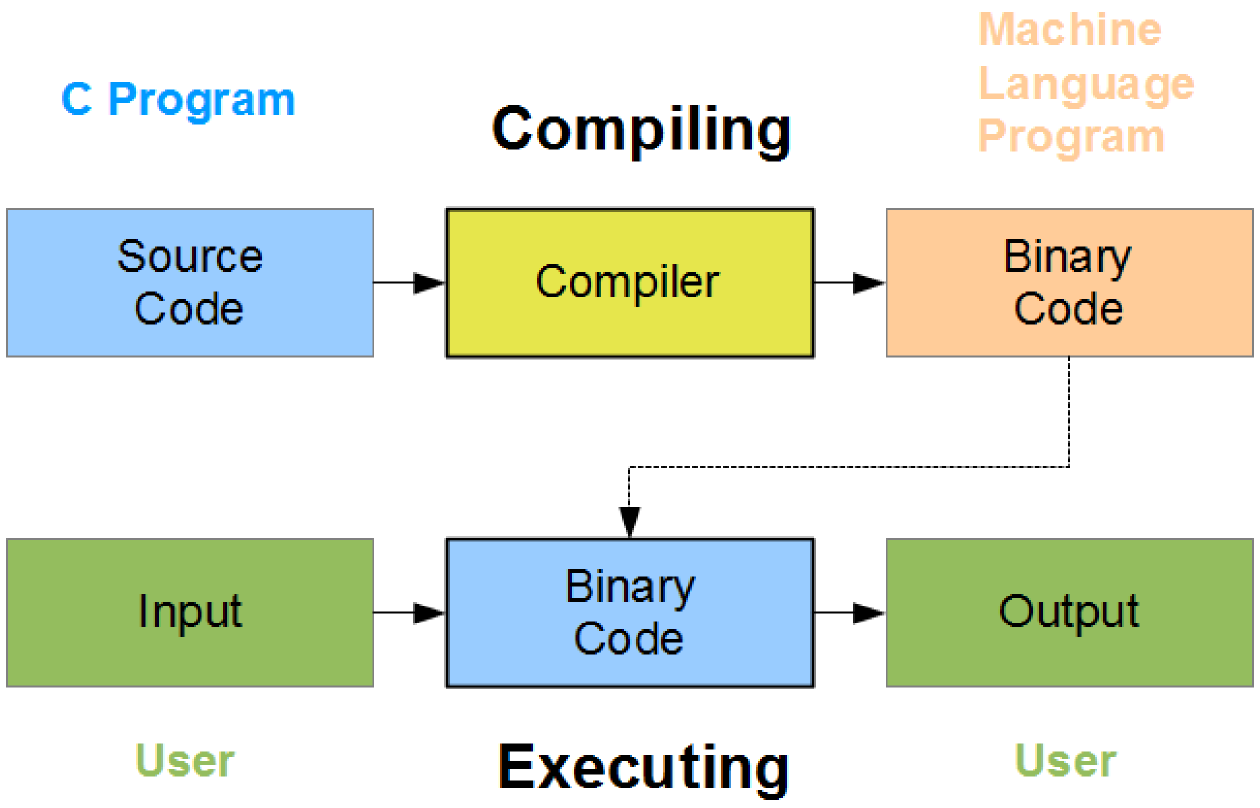
- To develop application software like database and spread sheets.
- UNIX Kernel is completely developed in C Language.
- C programming language can be used to design the compilers.

Features of C Language



How C Programming Language Works?

C is a compiled language. A compiler is a special tool that compiles the program and converts it into the object file which is machine readable. After the compilation process, the linker will combine different object files and creates a single executable file to run the program. The following diagram shows the execution of a 'C' program



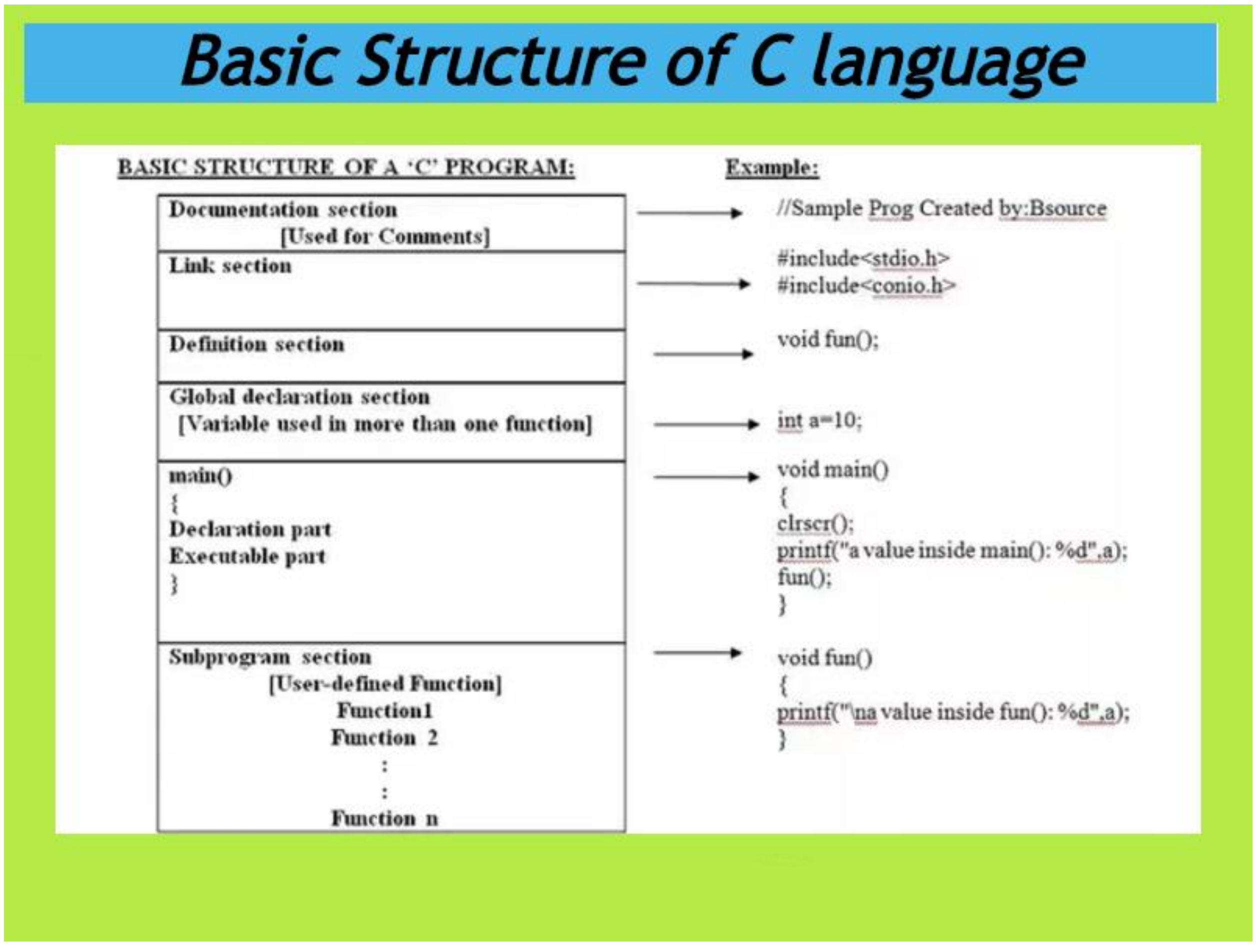
Nowadays, various compilers are available online, and you can use any of those compilers. The functionality will never differ and most of the compilers will provide the features required to execute both 'C' and 'C++' programs.

Following is the list of popular compilers available online:

- Clang compiler
- MinGW compiler (Minimalist GNU for Windows)Portable 'C' compiler
- Turbo C

2. Chapter – Getting Started

Structure of c program:



```
//Sample Prog Created by:Bsource
#include<stdio.h>
#include<conio.h>
void fun();
int a=10;
void main()
{
clrscr();
printf("a value inside main(): %d",a);
fun();
}
void fun()
{
printf("\na value inside fun(): %d",a);
}
```

The C program Contain the Two Section:

Include Section	# include is a pre-processor directive can be used to include all the predefined functions of given header files into current C program before compilation.

Main function	This is starting executable block of any program One C program can have maximum one main() the entire statements of given program can be executed through main(). Without main() function no C program will be executed.

C Programming Input Output (I/O):

C programming has several in-build library functions to perform input and output tasks.

printf():-

Function sends formatted output to the standard output (screen).

Example 1: C Output

```
#include <stdio.h>      //This is needed to run printf() function.

int main()

{

    printf("C Programming");

    return 0;

}
```

Displays The Content Inside Quotation

Output:

    C Programming

scanf():

function reads formatted input from standard input (keyboard)

Example 3: C Integer Input/Output

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

Format String For Integer

```
    int test;
    printf("Enter an integer: ");
    scanf("%d",&test);
    printf("Number = %d",test);
    return 0;
}
```

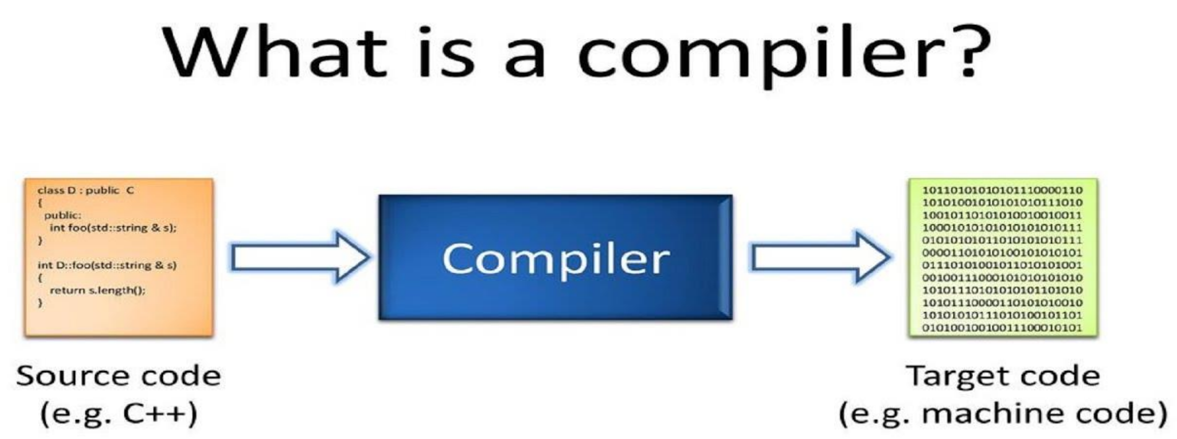
Output:

```
Enter an integer: 4

Number = 4
```

The **scanf()** function reads formatted input from the keyboard. When user enters an integer, it is stored in variable **test**. Note the **'&'** sign before **test**; **& test** gets the address of **test** and the value is stored in that address.

Compiler in C:

A compiler is system software which converts programming language code into binary format in single steps.



```
class D : public  C
{
 public:
  int foo(std::string & s);
}

int D::foo(std::string & s)
{
  return s.length();
}
```

```
10110101010101110000110
10101001010101010111010
10010110101010010010011
10001010101010101010111
01010101011010101010111
00001101010100101010101
01110101001011010101001
00100111000101010101010
10101110101010101101010
10101110000110101010010
10101010111010100101101
01010010010011100010101
```

Comments in C:

Generally **Comments** are used to provide the description about the Logic written in program. Comments are not display on output screen.

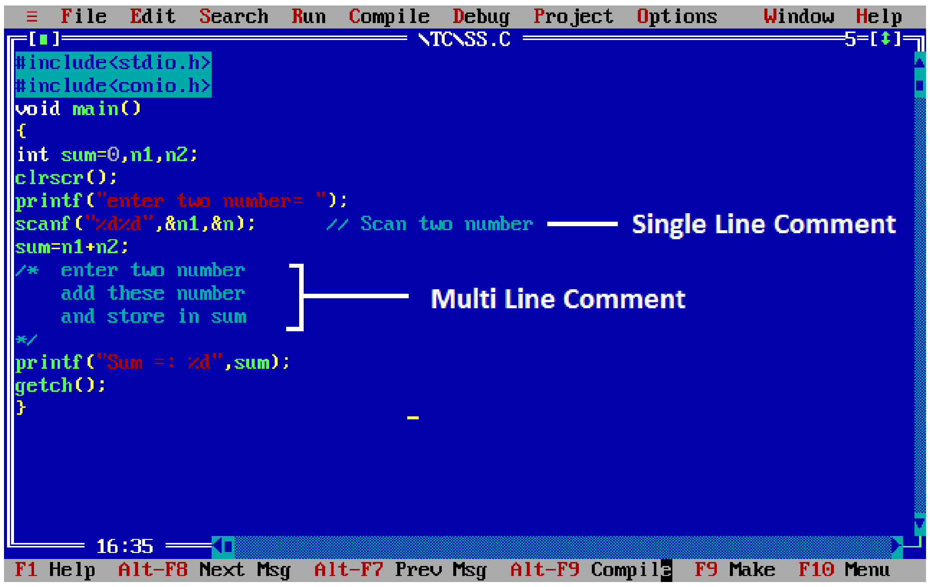
In 'C' language two types of comments are possible

Single line comments

Single line comments can be provided by using / /....................

Multiple line comments

Multiple line comments can be provided by using /*......................*/


≡ File Edit Search Run Compile Debug Project Options Window Help
\TC\SS.C
Single Line Comment
Multi Line Comment
16:35
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
void main()
{
int sum=0,n1,n2;
clrscr();
printf("enter two number= ");
scanf("%d%d",&n1,&n);      // Scan two number
sum=n1+n2;
/*  enter two number
    add these number
    and store in sum
*/
printf("Sum =: %d",sum);
getch();
}
```

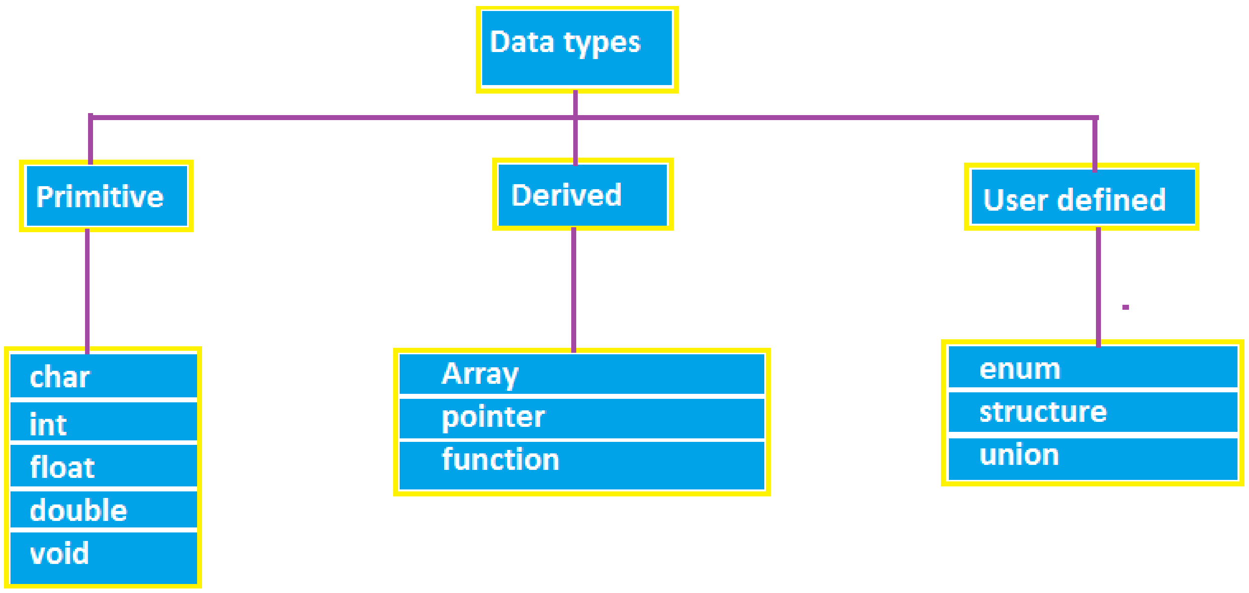
Keywords:

Keywords are the words whose meaning is already define in c compiler. Keywords are also called as **reserved words** which have special meaning .There are 32 keywords in c.

Auto	Double	Int	Struct
Break	Else	Long	Switch
Case	Enum	Register	Typedef
Char	Extern	Return	Union
Const	Float	Short	Unsigned
Continue	For	Signed	Void
Default	Goto	Sizeof	Volitle
Do	If	Static	While

Data Types

Data types simply refers to the type and size of data associated with variables and functions.The Following diagram shows types c Data types.

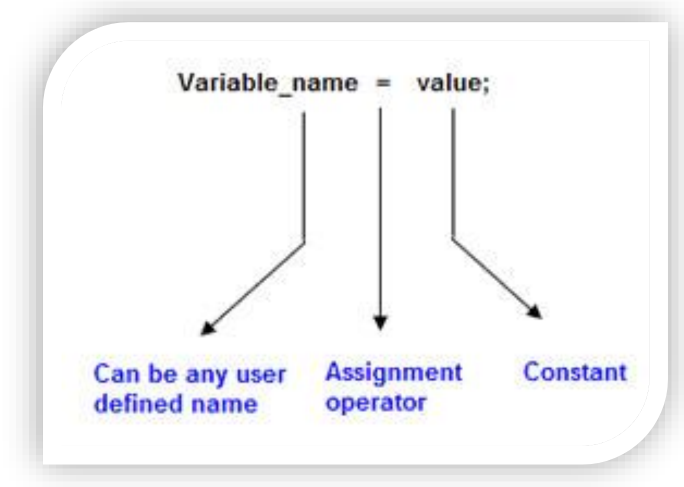


Data type	Storage Size	Value range	Example
Char	1 byte	-128 to 127	Char test='h';
Unsigned Char	1 byte	0 to 255	
Signed char	1 byte	-128 to 127	
Int	2 byte	-32,768 to 32,767	Int no=10;
Short int	2 byte	-32,768 to 32,767	
Long int	2 byte	-2,147,483,648 to 2,147,483,647	
Float	4 byte	1.2E-38 to 3.4E+38	float no=3.14;
Double	8 byte	2.3E-308 to 1.7E+308	

Variable:

Variable *is an identifier which holds data or another one variable. It is an identifier whose value can be changed at the execution time of program. It is used to identify input data in a program.*

Initialization Of Variable:



Example 2: C Integer Output

```
#include <stdio.h>
int main()
{
   int testInteger = 5;
   printf("Number = %d", testInteger);
   return 0;
}
```

Variable Declaration & Initialization

Output:

```
Number = 5
```

1	Write a C Program to Print Hello Word
2	What is Keyword, Variable?
3	Write a C Program to declaring Variable and Printing its Value

Chapter 3. Operators

An operator is a symbol which operates on a value or a variable. For example: + is an operator to perform addition's programming has wide range of operators to perform various operations. For better understanding of operators, these operators can be classified as:

Type of Operator	Symbolic Representation
Arithmetic Operators	+, -, *, /, %
Increment and decrement operators	++,--
Assignment Operators	=
Relational Operators	==, > , <, >=, <=, !=
Logical Operators	&&, ||, !
Bitwise operator	&, !
Comma Operator	,
Conditional Operator	?:

No	Programs
1	Write a C Program to Calculate Area and Circumference of Circle
2	Write a C Program to Calculate Area of Triangle
3	Write a C Program to Calculate Area of Equilateral Triangle
4	Write a C Program to Find out the Cube of Any no.
5	Write a C Program to Calculate Area of Circle
6	Write a C Program to Calculate Area of Rectangle
7	Write a C Program to Calculate Area of Square
8	Write a C Program to Add the Five subject of Marks And Calculate its total.

4. Chapter – Decision Statement

Decision Making Statement

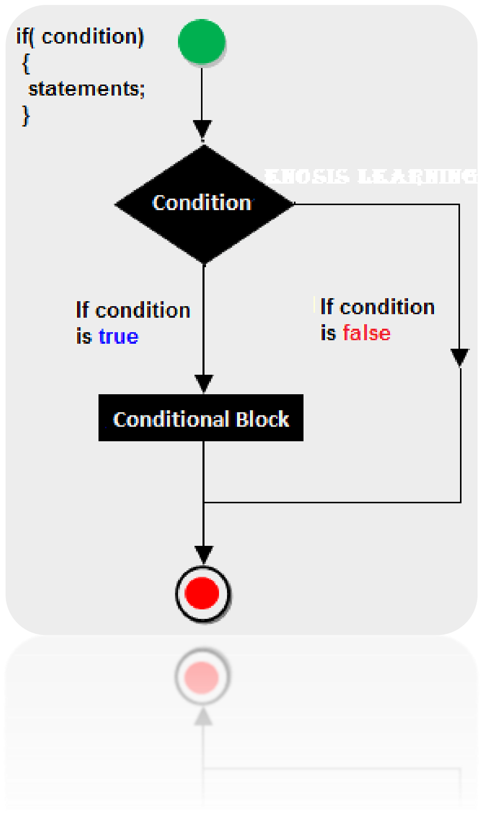
Decision making statement is depending on the condition block need to be executed or not which is decided by condition.

If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed.

- if
- if-else
- else if statement

1.if statement:

If test expression is evaluated to true (nonzero), statements inside the body of **if** is executed. If test expression is evaluated to false (0), statements inside the body of **if** are skipped. The Following Diagram Contains the Syntax And flowchart



Example 1: C if statement

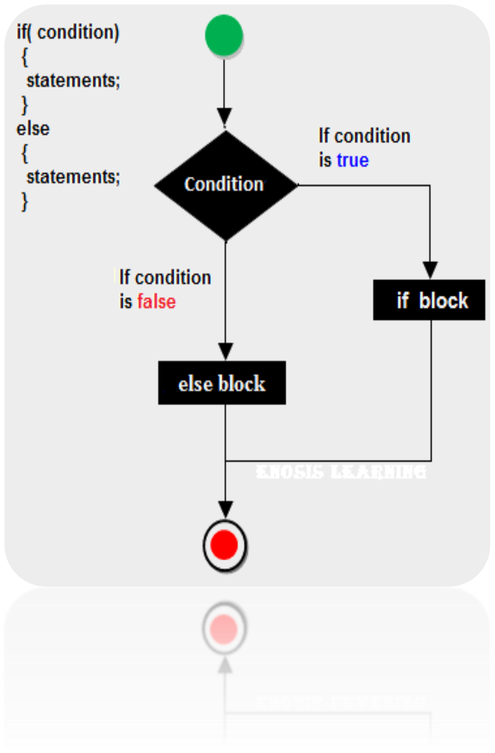
```
// Program to display a number if user enters negative
number

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

    printf("Enter an integer: ");
    scanf("%d", &number);

    // Test expression is true if number is less than 0
    if (number < 0)
    {
        printf("You entered %d.\n", number);
    }

    printf("The if statement is easy.");

    return 0;
}
```

Output :

```
Enter an integer: -2
You entered -2.
The if statement is easy.
```

2.if...else statement:

If test expression is true, code inside the body of **if** statement is executed; and code inside the body of **else** statement is skipped.
If test expression is false, code inside the body of **else** statement is executed; and code inside the body of **if** statement is skipped.



Example 2: C if...else statement

```
// Program to check whether an integer entered by the user is odd or even

#include <stdio.h>
int main()
{
   int number;
   printf("Enter an integer: ");
   scanf("%d",&number);

   // True if remainder is 0
   if( number%2 == 0 )
      printf("%d is an even integer.",number);
   else
      printf("%d is an odd integer.",number);
   return 0;
}
```

Output

```
Enter an integer: 7

7 is an odd integer.
```

3.Nested if...else statement (if...else if....else Statement):

The **if...else** statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.
The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.

Syntax of nested if...else statement:

```
if (testExpression1)
{
  // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
  // statements to be executed if testExpression1 is false and testExpression2 is true
}
else if (test Expression 3)
{
  // statements to be executed if testExpression1 and testExpression2 is false and
testExpression3 is true
}
.
.
else
```

```
{
  // statements to be executed if all test expressions are false
}
```

Example 3: C nested if...else statement

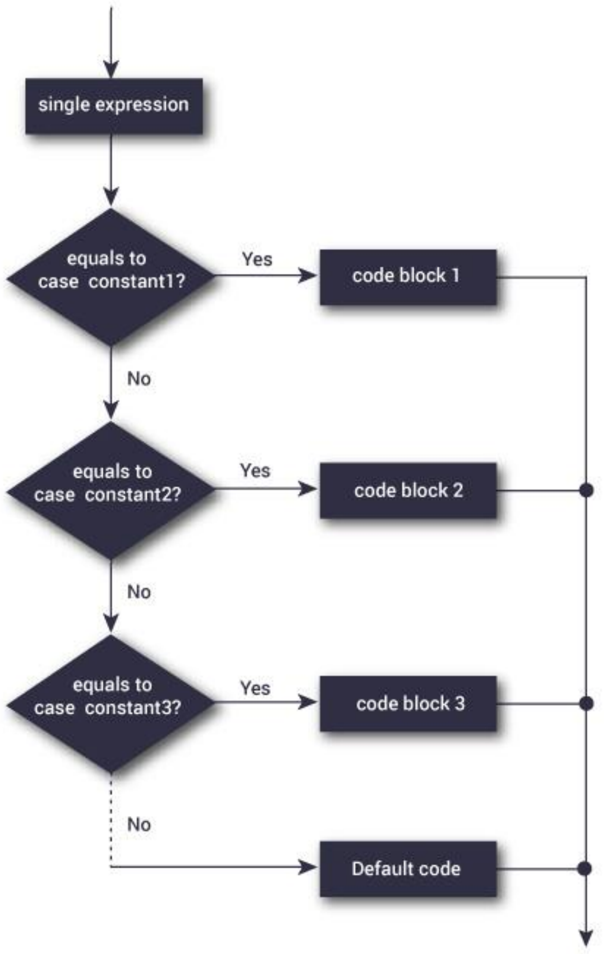
```
// Program to relate two integers using =, > or <
#include <stdio.h>
int main()
{
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    //checks if two integers are equal.
    if(number1 == number2)
    {
        printf("Result: %d = %d",number1,number2);
    }

    //checks if number1 is greater than number2.
    else if (number1 > number2)
    {
        printf("Result: %d > %d", number1, number2);
    }

    // if both test expression is false
    else
    {
        printf("Result: %d < %d",number1, number2);
    }

    return 0;
}
```

Output

```
Enter two integers: 12

23

Result: 12 < 23
```

switch...case Statement:

The nested **if...else** statement allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in nested if...else statement, it is better to use **switch** statement. The switch statement is often faster than nested **if...else** (not always). Also, the syntax of switch



Syntax of switch...case:

```
switch (n)
{
case constant1:
          // code to be executed i
break;
case constant2:
             // code to be executed if n is equal to constant2;
break;
.
.
default:
             // code to be executed if n doesn't match any constant
}
```

N Should Be Expression

Example: switch Statement

```
#include <stdio.h>
int main ()
{
   int value = 3;
   switch(value)
   {
     case 1:
     printf("Value is 1 \n" );
     break;

     case 2:
     printf("Value is 2 \n" );
     break;
     case 3:
     printf("Value is 3 \n" );
     break;
     case 4:
     printf("Value is 4 \n" );
     break;
     default :
     printf("Value is other than 1,2,3,4 \n" );
   }
```

```
  return 0;
}
```

Output:

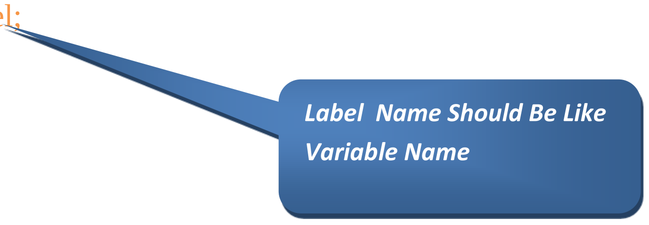
```
 Value is 3
```

C Programming go to Statement:

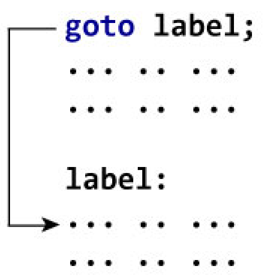
The goto statement is used to alter the normal sequence of a C program.

Syntax of goto statement:

```
goto label;
... .. ...
... .. ...
... .. ...
label:
statement;
```



The label is an identifier. When **goto** statement is encountered, control of the program jumps to**label:** and starts executing the code.



```
goto label;
... .. ...
... .. ...

label:
... .. ...
... .. ...
```

Example: goto Statement

```
#include <stdio.h>
int main()
{
   int i;
   for(i=0;i<10;i++)
     {
   if(i==5)
   {
     printf("\nWe are using goto statement when i = 5");
     goto HAI;
   }
   printf("%d ",i);
}
HAI : printf("\nNow, we are inside label name \"hai\" \n");
}
```

Output

```
0 1 2 3 4

We are using goto statement when i = 5

Now, we are inside label name “hai”
```

Reasons to avoid goto statement

The use of goto statement may lead to code that is buggy and hard to follow. For example:

```
one:
for (i = 0; i < number; ++i)
{
    test += i;
    goto two;
}
two:
if (test > 5) {
  goto three;
}
```

```
... .. ...
```

Also, goto statement allows you to do bad stuff such as jump out of scope.

Chapter 4-Exercise(Decision Statements Programs)

No	Programs
1	Write a C program to find maximum between two numbers.
2	Write a C program to find maximum between three numbers.
3	Write a C program to check whether a number is negative, positive or zero.
4	Write a C program to check whether a number is divisible by 5 and 11 or not.
5	Write a C program to check whether a number is even or odd.
6	Write a C program to check whether a year is leap year or not.
7	Write a C program to check whether a character is alphabet or not.
8	Write a C program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to following: Percentage >= 90% : Grade A Percentage >= 80% : Grade B Percentage >= 70% : Grade C Percentage >= 60% : Grade D Percentage >= 40% : Grade E Percentage < 40% : Grade F
9	Write a C program to input any alphabet and check whether it is vowel or consonant.

5. Chapter – Loop Statement

C Programming Loop:

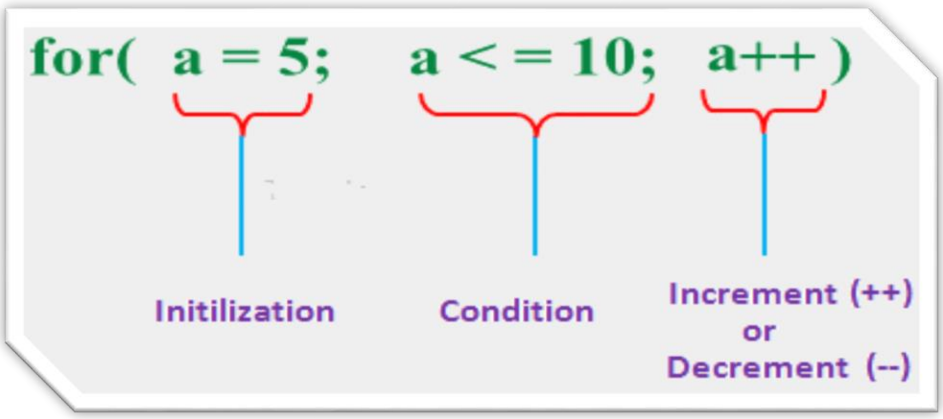
Loops are used in programming to repeat a specific block of code.

Loops are used in programming to repeat a specific block until some end condition is met. There are three loops in C programming:

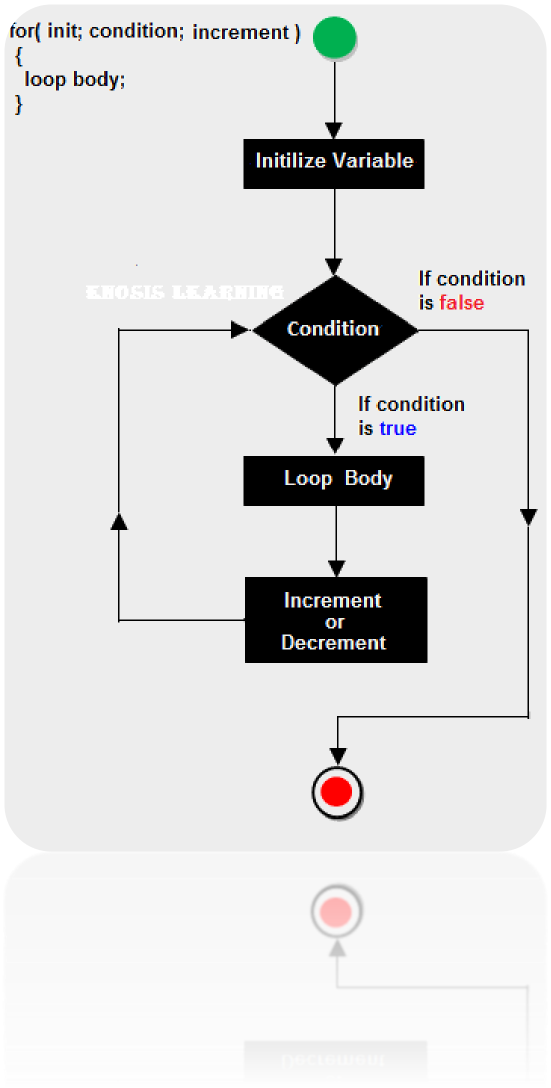
1. for loop
2. while loop
3. do...while loop

1.For Loop:

The Following Diagram Contain the Syntax:



Following Chart Contain the Flowchart Of For Loop how Execute the For Loop-



Example: for loop

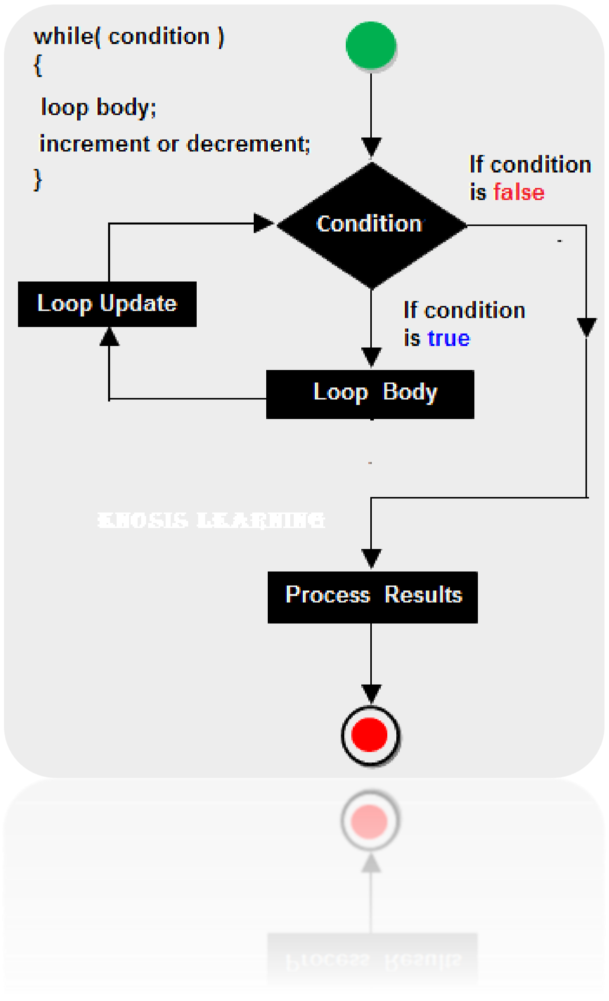
```
#include <stdio.h>
 int main ()
 {
   int a;
   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 )
{
      printf("value of a: %d\n", a);
   }
   return 0;
}
```

Output:-

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

While Loop:

In **while loop** First check the condition if condition is true then control goes inside the loop body other wise goes outside the body. **while loop** will be repeats in clock wise direction.



Example 1: while loop

```
#include <stdio.h>
 int main () {
   /* local variable definition */
   int a = 10;
   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }
   return 0;
}
```

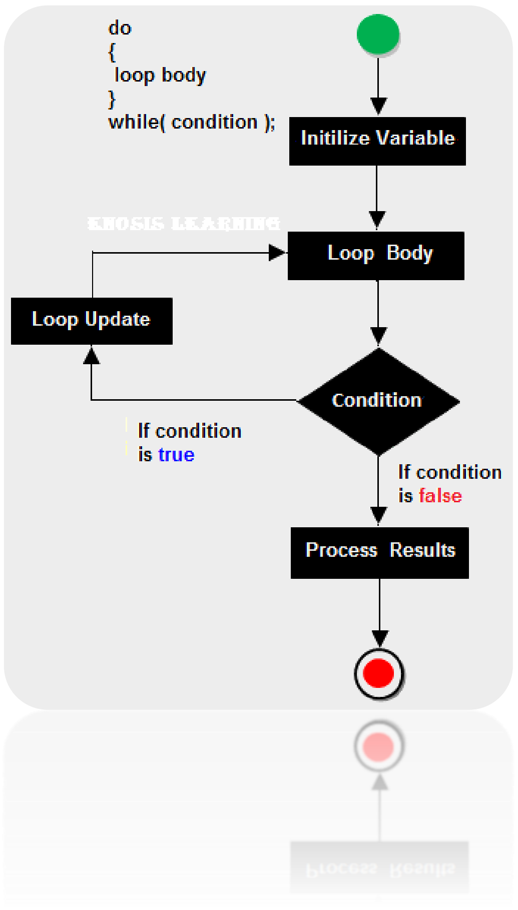
Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

2.do...while loop:

A **do-while** loop is similar to a **while** loop, except that a do-while loop is execute at least one time.

A **do while** loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given condition at the end of the block (in while).



Example 2: do...while loop

```
#include <stdio.h>
 int main ()
{
   /* local variable definition */
   int a = 10;
   /* do loop execution */
   do {
      printf("value of a: %d\n", a);
      a = a + 1;
   }while( a < 20 );
    return 0;
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a:19
```

C Programming break and continue Statement:

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, **break** and **continue** statements are used.

break Statement:

The break statement terminates the loop immediately when it is encountered. The break statement is used with decision making statement such as if...else.

Syntax of break statement

Break;

Example: break statement

```
#include <stdio.h>
int main()
{
   int i;
  for(i=0;i<10;i++)
   {
     if(i==5)
     {
        printf("\nComing out of for loop when i = 5");
        break;
     }
     printf("%d ",i);
   }
}
```

Output

```
 0 1 2 3 4

 Coming out of for loop when i = 5
```

In C programming, break statement is also used with switch...case statement.

continue Statement:

The continue statement skips some statements inside the loop. The continue statement is used with decision making statement such as if...else.

Syntax of continue Statement:

```
continue;
```

Example : continue statement

```
#include <stdio.h>
int main()
{
   int i;
   for(i=0;i<10;i++)
   {
     if(i==5 || i==6)
     {
       printf("\skipping %d from display using " \
       "continue statement \n",i);
       continue;
     }
     printf("%d ",i);
   }
}
```

Output

```
0 1 2 3 4

Skipping 5 from display using continue statement

Skipping 6 from display using continue statement

7 8 9
```

Chapter 5-Exercise (loop based programs)

1	Write a C program to print all natural numbers from 1 to n. - using while loop
2	Write a C program to print all natural numbers in reverse (from n to 1). - using while loop
3	Write a C program to print all alphabets from a to z. - using while loop
4	Write a C program to print all even numbers between 1 to 100. - using while loop
5	Write a C program to print all odd number between 1 to 100.
6	Write a C program to find sum of all natural numbers between 1 to n.
7	Write a C program to count number of digits in any number.
8	Write a C program to find HCF (GCD) of two numbers.
9	Write a C program to find LCM of two numbers.

10	Write a C program to check whether a number is Prime number or not.

Chapter 6: Functions

A **function** is a group of statements that together perform a specific task. Every C program has at least one function, which is main().

Why use function ?

Function are used for **divide a large code into module**, due to this we can easily debug and maintain the code. For example if we write a calculator programs at that time we can write every logic in a separate function (For addition sum(), for subtraction sub()). Any function can be called many times.

Advantage of Function

- Code Re-usability
- Develop an application in module format.

- Easily to debug the program.
- **Code optimization:** No need to write lot of code.

Types of functions:

Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming

There are two types of functions in C programming:

- Standard library functions
- User defined functions

Standard library functions:

The standard library functions are in-built functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.

The **printf()** is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in **"stdio.h"** header file.

There are other numerous library functions defined under **"stdio.h"**, such as **scanf()**, **fprintf()**,**getchar()** etc. Once you include **"stdio.h"** in your program, all these functions are available for use.

User defined function:

These functions are created by programmer according to their requirement for example suppose you want to create a function for add two number then you create a function with name sum() this type of function is called user defined function.

Function Declarations

A function declaration is the process of tells the compiler about a function name. The actual body of the function can be defined separately.

Syntax:

```
return_type  function_name(parameter);
```

Note: At the time of function declaration function must be terminated with **;**.

Defining a function.

Defining of function is nothing but give body of function that means write logic inside function body.

Syntax:

```
return_type  function_name(parameter)
{
function body;
}
```

- **Return type:** A function may return a value. The return_type is the data type of the value the function returns.Return type parameters and returns statement are optional.
- **Function name:** Function name is the name of function it is decided by programmer or you.

- **Parameters:** This is a value which is pass in function at the time of calling of function A parameter is like a placeholder. It is optional.
- **Function body:** Function body is the collection of statements.

calling a function.

When we call any function control goes to function body and execute entire code. For call any function just write name of function and if any parameter is required then pass parameter.

Syntax:

```
Function_name();
```

Example: User-defined function

Here is a example to add two integers. To perform this task, a user-defined function **addNumbers()** is defined.

```
#include <stdio.h>
int addNumbers(int a, int b);
int main()
{
    int n1,n2,sum;
    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);
    printf("sum = %d",sum);
    return 0;
}
int addNumbers(int a,int b)        // function definition
{
    int result;
    result = a+b;
    return result;                 // return statement
}
```

Function Declaration

Function Call

Types Of User Defined Functions:

For better understanding of arguments and return value from the function, user-defined functions can be categorized as:

- **Function with no arguments and no return value**
- **Function with no arguments and a return value**
- **Function with arguments and no return value**
- **Function with arguments and a return value.**

The following programs below write a program to print an sum of two numbers. And, all these programs generate the same output.

Example 1: No arguments passed and no return Value

```
#include <stdio.h>
void sum();
int main()
{
   sum();    // no argument is passed to sum()
   return 0;
}
// return type of the function is void becuase no value is returned from the
function
void sum()
{
   int a, b, sum;
   printf("Enter a first no: ");
   scanf("%d",&a);
printf("Enter a second no: ");
   scanf("%d",&b);
sum=a+b;
printf(“Sum=%d”,sum);
}
```

The empty parentheses in **sum();** statement inside the **main()** function indicates that no argument is passed to the function.

The return type of the function is **void**. Hence, no value is returned from the function.

The **sum()** function takes input from the user, print the addition of two numbers.

Example 2: No arguments passed but a return value

```
#include <stdio.h>
int getInteger();
int main()
{
    int n, i, flag = 0;

    // no argument is passed to the function
    // the value returned from the function is assigned to n
    n = getInteger();
    for(i=2; i<=n/2; ++i)
    {
        if(n%i==0){
            flag = 1;
            break;
        }
    }
    if (flag == 1)
        printf("%d is not a prime number.", n);
    else
        printf("%d is a prime number.", n);
    return 0;
}
// getInteger() function returns integer entered by the user
int getInteger()
{
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    return n;
}
```

The empty parentheses in **n = getInteger();** statement indicates that no argument is passed to the function. And, the value returned from the function is assigned to **n**.

Here, the **getInteger()** function takes input from the user and returns it. The code to check whether a number is prime or not is inside the **main()** function.

Example3: Argument passed but no return value

```
#include <stdio.h>
void sum(int a,int b);
int main()
{
 int a, b;
  printf("Enter a first no: ");
  scanf("%d",&a);
  printf("Enter a second no: ");
  scanf("%d",&b);
   sum(a,b);
  return 0;
}
void sum(int a,int b)
{
int res;
res=a+b;
printf("Sum=%d",res);
}
```

The integer value entered by the user is passed to **sum()** function.

Here, the **sum()** function checks whether the argument passed is a prime number or not and displays the appropriate message.

Example 4: Argument passed and a return value

```
#include <stdio.h>
int sum(int a,int b);
int main()
{
 int a, b,res;
   printf("Enter a first no: ");
    scanf("%d",&a);
printf("Enter a second no: ");
    scanf("%d",&b);
   res= sum(a,b);
                printf("Sum=%d",res);
            return 0;
}
int sum(int a,int b)
{
int res;
res=a+b;
return res;
}
```

The input from the user is passed to **sum()** function.

C Programming Recursion:

A function that calls itself is known as recursive function. And, this technique is known as recursion.

How recursion works?

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

Example: Sum of Natural Numbers Using Recursion

```
#include <stdio.h>
int sum(int n);
int main()
{
    int number, result;
    printf("Enter a positive integer: ");
    scanf("%d", &number);
    result = sum(number);
    printf("sum=%d", result);
}
int sum(int n)
{
    if (n!=0)
```

```
        return n + sum(n-1); // sum() function calls itself
    else
        return n;
}
```

Output

```
Enter a positive integer:
3
6
```

Initially, the **sum()** is called from the **main()** function with **number** passed as an argument.

Suppose, the value of **n** is 3 initially. During next function call, 2 is passed to the **sum()** function. In next function call, 1 is passed to the function. This process continues until **n** is equal to 0.

When **n** is equal to 0, there is no recursive call and the sum of integers is returned to the **main()**function.

Advantages and Disadvantages of Recursion:

Recursion makes program elegant and cleaner. All algorithms can be defined recursively which makes it easier to visualize and prove.

If the speed of the program is vital then, you should avoid using recursion. Recursions use more memory and are generally slow.

Storage Class:

storage class determines the scope and lifetime of a variable.

There are 4 types of storage class:

1. automatic
2. external

3. static
4. register

Local Variable

The variables declared inside the function are automatic or local variables.

The local variables exist only inside the function in which it is declared. When the function exits, the local variables are destroyed.

```
int main() {

    int n;

    ... .. ...

}
```

Local Variable

```
void func() {

   int n1; // n1 is local to func() fucntion

}
```

Global Variable:

Variables that are declared outside of all functions are known as external variables. External or global variables are accessible to any function.

Example 1: External Variable

```
#include <stdio.h>
void display();
int n = 5;
int main()
{
    ++n;     // variable n is not declared in the main() function
    display();
```

Global Variable

```
    return 0;
 }
 void display()
 {
    ++n;     // variable n is not declared in the display() function
    printf("n = %d", n);
 }
```

Output:

```
        n=7
```

Suppose, a global variable is declared in **file1**. If you try to use that variable in a different file**file2**, the compiler will complain. To solve this problem, keyword **extern** is used in **file2** to indicate that the external variable is declared in another file.

Auto Variable:

The **auto** storage class is the default storage class for all local variables. The scope **auto** variable is within the function. It is equivalent to local variable.

Syntax:

```
Auto int a;
```

Register Variable:

The **register** keyword is used to declare register variables. Register variables were supposed to be faster than local variables.

However, modern compilers are very good at code optimization and there is a rare chance that using register variables will make your program faster.

Unless you are working on embedded system where you know how to optimize code for the given application, there is no use of register variables.

Static Variable:

A static variable is declared by using keyword **static**. For example;

```
static int i;
```

The value of a static variable persists until the end of the program.

Example 2: Static Variable

```
#include <stdio.h>
void display();
int main()
{
   display();
   display();
}
void display()
{
   static int c = 0;
   printf("%d  ",c);
   c += 5;
}
```

Output

```
0 5
```

During the first function call, the value of **c** is equal to 0. Then, it's value is increased by 5.

During the second function call, variable **c** is not initialized to 0 again. It's because **c** is a static variable. So, 5 is displayed on the screen.

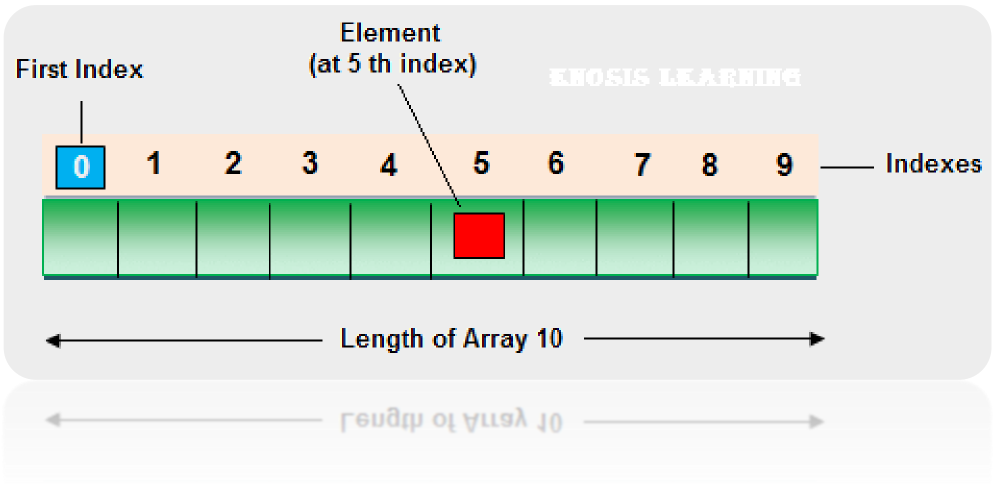
1	Write a C program to find cube of any number using function. .
2	Write a C program to find diameter, circumference and area of circle using functions.
3	Write a C program to find maximum and minimum between two numbers using functions.
4	Write a C program to check whether a number is even or odd using functions.
5	Write a C program to check whether a number is prime, Armstrong or perfect number using functions.
6	Write a C program to find all prime numbers between given interval using functions.
7	Write a C program to print all strong numbers between given interval using functions.
8	Write a C program to print all Armstrong numbers between given interval using functions.
9	Write a C program to print all perfect numbers between given interval using functions

Chapter 7:Array

Array in C Language:

An array is a collection of similar data type value in a single variable. It is a derived data type in C, which is constructed from fundamental data type of C language.

How Array Works :



An array is a sequence of data item of homogeneous value(same type).

Arrays are of two types:

1. One-dimensional arrays
2. Multidimensional arrays

1) One Dimensional arrays:

Syntax of one dimensional array is as follows:

data_type array_name[array_size];

Initialization of one-dimensional array:

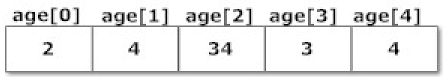
Arrays can be initialized at declaration time in this source code as:

```
int age[5]={2,4,34,3,4};
```

It is not necessary to define the size of arrays during initialization.

```
int age[]={2,4,34,3,4};
```

In this case, the compiler determines the size of array by calculating the number of elements of an array.



Initialization of one-dimensional array

Accessing array elements:

```
/* C program to find the sum marks of n students using arrays */
#include <stdio.h>
int main(){
     int marks[10],i,n,sum=0;
     printf("Enter number of students: ");
     scanf("%d",&n);
     for(i=0;i<n;++i)
{
          printf("Enter marks of student%d: ",i+1);
          scanf("%d",&marks[i]);
          sum+=marks[i];
     }
     printf("Sum= %d",sum);
return 0;
}
```

Accepting The Array element

Output

```
Enter number of students: 3
Enter marks of student1: 12
Enter marks of student2: 31
Enter marks of student3: 2
sum=45
```

Two Dimensional Arrays

The Syntax of two dimensional array as define is as follws:

datatype arrayname[size][size];

Example of Two Dimensional Array In C

Write a C program to find sum of two matrix of order 2*2 using multidimensional arrays where, elements of matrix are entered by user.

```
#include <stdio.h>

int main()
{
  int m, n, c, d, first[10][10], second[10][10], sum[10][10];

  printf("Enter the number of rows and columns of matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");
  for (c = 0; c < m; c++)
    for (d = 0; d < n; d++)
      scanf("%d", &first[c][d]);
  printf("Enter the elements of second matrix\n");
  for (c = 0; c < m; c++)
    for (d = 0 ; d < n; d++)
        scanf("%d", &second[c][d]);

  printf("Sum of entered matrices:-\n");

  for (c = 0; c < m; c++) {
    for (d = 0 ; d < n; d++) {
      sum[c][d] = first[c][d] + second[c][d];
      printf("%d\t", sum[c][d]);
    }
    printf("\n");
  }
  return 0;
}
```

Ouput:

```
Enter the number of rows and columns of matrix
2
2
First Matrix :-
1 2
3 4
```

```
Second matrix :-
4 5
-1 5
sum of entered matrix is:
5 7
2 9
```

C Programming Arrays and Functions:

In C programming, a single array element or an entire array can be passed to a function. Also, both one-dimensional and multi-dimensional array can be passed to function as argument.

Passing One-dimensional Array In Function

C program to pass a single element of an array to function

```
#include <stdio.h>
void display(int a)
   {
   printf("%d",a);
   }
int main(){
   int c[]={2,3,4};
   display(c[2]);  //Passing array element c[2] only.
   return 0;
}
```

Output

```
4
```

Single element of an array can be passed in similar manner as passing variable to a function.

Passing entire one-dimensional array to a function

While passing arrays to the argument, the name of the array is passed as an argument(,i.e, starting address of memory area is passed as argument).

```
#include<stdio.h>
#include<conio.h>
//----------------------------------
void fun(int arr[])
{
int i;
for(i=0;i< 5;i++)
 arr[i] = arr[i] + 10;
}
//--------------------------------
void main()
{
int arr[5],i;
clrscr();
printf("\nEnter the array elements : ");
for(i=0;i< 5;i++)
 scanf("%d",&arr[i]);

printf("\nPassing entire array .....");
fun(arr);  // Pass only name of array

for(i=0;i< 5;i++)
 printf("\nAfter Function call a[%d] : %d",i,arr[i]);

getch();
}
```

Output

```
Enter the array elements : 1 2 3 4 5
Passing entire array .....
After Function call a[0] : 11
After Function call a[1] : 12
After Function call a[2] : 13
After Function call a[3] : 14
After Function call a[4] : 15
```

Passing Multi-dimensional Arrays to Function

To pass two-dimensional array to a function as an argument, starting address of memory area reserved is passed as in one dimensional array

Example to pass two-dimensional arrays to function

```
#include<stdio.h>
void Function(int c[2][2]);
int main(){
   int c[2][2],i,j;
   printf("Enter 4 numbers:\n");
   for(i=0;i<2;++i)
      for(j=0;j<2;++j){
           scanf("%d",&c[i][j]);
      }
   Function(c);   /* passing multi-dimensional array to function */
   return 0;
 }
 void Function(int c[2][2]){
 /* Instead to above line, void Function(int c[][2]){ is also valid */
   int i,j;
   printf("Displaying:\n");
   for(i=0;i<2;++i)
     for(j=0;j<2;++j)
        printf("%d\n",c[i][j]);
 }
```

Output

```
Enter 4 numbers:
2
3
4
5
Displaying:
2
3
4
5
```

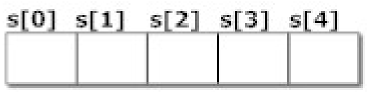
String:

String is a collection of character or group of character, it is achieve in C language by using array character. The string in C language is one-dimensional array of character which is terminated by a null character '\0'. In other words string is a collection of character which is enclose between double cotes (" ").

Declaration of strings:

Strings are declared in C in similar manner as arrays. Only difference is that, strings are of **char** type.

```
char s[5];
```



Initialization of strings:

In C, string can be initialized in different number of ways.

char c[]="abcd";

OR,

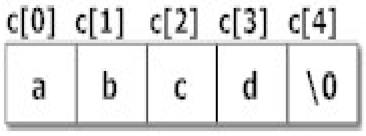
char c[5]="abcd";

OR,

char c[]={'a','b','c','d','\0'};

OR;

char c[5]={'a','b','c','d','\0'};



Reading Strings from user.

```
char c[20];
scanf("%s",c);
```

String variable **c** can only take a word. It is beacause when white space is encountered, the **scanf()** function terminates.

Write a C program to illustrate how to read string from terminal.

```
#include <stdio.h>
int main(){
    char name[20];
    printf("Enter name: ");
    scanf("%s",name);
    printf("Your name is %s.",name);
    return 0;
}
```

Output:

```
Enter name: Dennis Ritchie

Your name is Dennis.
```

Here, program will ignore Ritchie because, **scanf()** function takes only string before the white space.

Reading a line of text:

This process to take string is tedious. There are predefined functions **gets()** and **puts** in C language to read and display string respectively.

```
int main(){
    char name[30];
    printf("Enter name: ");
    gets(name);     //Function to read string from user.
    printf("Name: ");
    puts(name);    //Function to display string.
    return 0;
}
```

Both, the above program has same output below:

Output:

```
Enter name: Tom Hanks
Name: Tom Hanks
```

Passing Strings to Functions:

String can be passed to function in similar manner as arrays as, string is also an array.

```
#include<stdio.h>
void Display(char ch[]);
int main(){
    char c[50];
    printf("Enter string: ");
    gets(c);
    Display(c);     // Passing string c to function.
    return 0;
}
void Display(char ch[]){
    printf("String Output: ");
    puts(ch);}
```

Here, string **c** is passed from **main()** function to user-defined function **Display()**. In function declaration, **ch[]** is the formal argument.

String Manipulations Functions:

There are numerous functions defined in **"string.h"** header file. Few commonly used string handling functions are discussed below:

Function	Work of Function
strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string
strlwr()	Converts string to lowercase
strupr()	Converts string to uppercase

Chapter 7: Exercise(Array & String)

Chapter 7: Exercise(Array & String)

1	Write a C program to read and print elements of array. - using recursion.
2	Write a C program to print all negative elements in an array.
3	Write a C program to find sum of all array elements. - using recursion.
4	Write a C program to find maximum and minimum element in an array. - using recursion.
5	Write a C program to find second largest element in an array.
6	Write a C program to count total number of even and odd elements in an array.
7	Write a C program to count total number of negative elements in an array
8	Write a C program to find length of a string.
9	Write a C program to copy one string to another string.
10	Write a C program to concatenate two strings.
11	Write a C program to compare two strings.
12	Write a C program to convert lowercase string to uppercase. Write a C program to convert uppercase string to lowercase

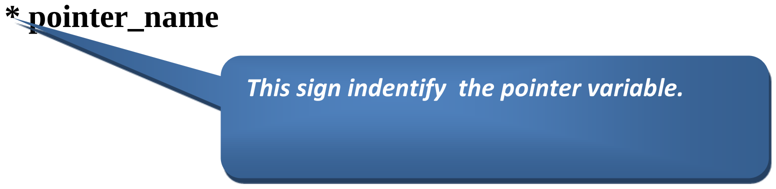
Chapter 8: Pointer, Structure &union

Pointer:

A **pointer** is a variable which contains or hold the address of another variable. We can create pointer variable of any type of variable for example integer type pointer is

Syntax:

Datatype * pointer_name



Symbol	Name	Description
& (ampersand sign)	Address of operator	Give the address of a variable
* (asterisk sign)	Indirection operator	Gives the contents of an object pointed to by a pointer.

Advantage of pointer:

- Pointer reduces the code and improves the performance, because it direct access the address of variable.
- Using pointer concept we can return multiple value from any function.
- Using pointer we can access any memory location from pointer.

Declaring a pointer

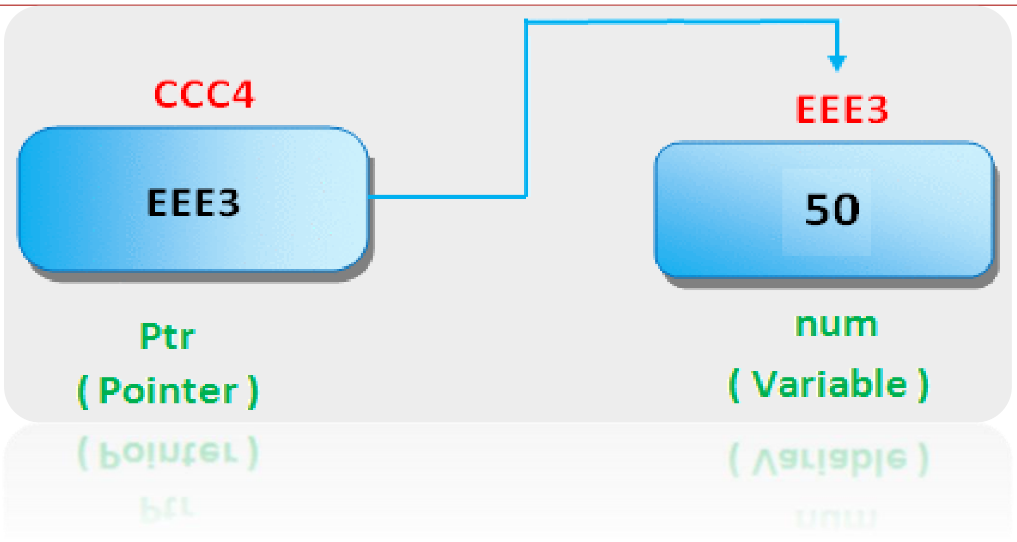
In C language for declared pointer we can use *** (asterisk symbol)**.

Syntax

```
int *p;  //pointer to integer
char *ch; //pointer to character
```

Example of pointer

In below image pointer variable stores the address of num variable i.e. EEE3. The value of num is 50 and address of pointer prt is CCC4



Example Of Pointer:

```
#include<stdio.h>
Int main()
{
  Int no=50;
  Int *ptr;
  Ptr=&no;
  Printf("Address of num variable",&num)
  Printf("address of  stored in ptr variable",ptr)
 Printf("value Of *ptr variable",*ptr);
}
```

Structure in C:

Structure is a user defined data type which hold or store heterogeneous data item Structure is the collection of variables of different types under a single name for better handling.

Why Use Structure in C

In C language array is also a user defined data type but array hold or store only similar type of data, If we want to store different-different type of data in then we need to defined separate variable for each type of data.

Example: Suppose we want to store Student record, then we need to store....

- Student Name
- Roll number
- Class
- Address

For store Student name and Address we need character data type, for Roll number and class we need integer data type. In this case we define the structure.

Structure Definition in C

Keyword **struct** is used for creating a structure.

Syntax of structure

```
struct structure_name

{

    data_type member1;
    data_type member2;

    .

    .

    data_type memeber;

};
```

We can create the structure for a person as mentioned above as:

```
struct person
{
    char name[50];
    int cit_no;
    float salary;
};
```

This declaration above creates the derived data type **struct person**.

Structure variable declaration:

When a structure is defined, it creates a user-defined type but, no storage is allocated. For the

```
struct person
{
    char name[50];
    int cit_no;
    float salary;
};
Inside main function:
struct person p1, p2, p[20];
```

Another way of creating sturcture variable is:

```
struct person
{
    char name[50];
    int cit_no;
    float salary;
}p1 ,p2 ,p[20];
```

Structure Variable Declaration/Objects

In both cases, 2 variables **p1**, **p2** and array **p** having 20 elements of type **struct person** are created.

Accessing members of a structure

There are two types of operators used for accessing members of a structure.

1. Member operator(.)
2. Structure pointer operator(->) (will be discussed in structure and pointers chapter)

Any member of a structure can be accessed as: **structure_variable_name.member_name**

Suppose, we want to access salary for variable **p2**. Then, it can be accessed as:

```
p2.salary
```

Example of structure

```
#include <stdio.h>
#include <string.h>
struct student
{
            int id;
            char name[20];
            float percentage;
};
int main()
{
            struct student record;
           Printf(“Enter id ane name ane per”);
           Scanf(“%d%s%f”,&record.id,record.name,& record.percentage);
            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
            return 0;
}
```

Member Operator For accessing the variable define in the structure

Output:

```
Enter id ane name ane per:
1
Raju
86.5
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Array Of Structure:

Structure is collection of different datatypes (variables) which are grouped together. Whereas, array of structures is nothing but collection of structures. This is also called as structure array in C.

EXAMPLE OR ARRAY OF STRUCTURES IN C:

This program is used to store and access “id, name and percentage” for 3 students

```
#include <stdio.h>
#include <string.h>
struct student
{
    int id;
    char name[30];
    float percentage;
};
int main()
{
    int i;
    struct student record[2];
    for(i=0;i<3;i++)
{
printf(“Enter roll no”);
scanf(“%d”,&record[i].id);
printf(“Enter name”);
scanf(“%d”,&record[i].name);
printf(“Enter percentage”);
scanf(“%d”,&record[i].percentage)
}
    for(i=0; i<3; i++)
    {
       printf("     Records of STUDENT : %d \n", i+1);
       printf(" Id is: %d \n", record[i].id);
       printf(" Name is: %s \n", record[i].name);
       printf(" Percentage is: %f\n\n",record[i].percentage);
    }
    return 0;
}
```

Output:

```
Enter rollno:1

Enter name:arti

Enter percentage:88.66

Enter rollno:2

Enter name:akshay

Enter percentage:77.66

Enter rollno:3

Enter name:smita

Enter percentage:88.66

Records of STUDENT : 1
Id is: 1
Name is: Raju
Percentage is: 86.500000

Records of STUDENT : 2
Id is: 2
Name is: Surendren
Percentage is: 90.500000

Records of STUDENT : 3
Id is: 3
Name is: Thiyagu
Percentage is: 81.500000
```

Structure using Function:

A structure variable can be passed to the function as an argument as normal variable. If structure is passed by value, change made in structure variable in function definition does not reflect in original structure variable in calling function.

Write a C program to create a structure student, containing name and roll. Ask user the name and roll of a student in main function. Pass this structure to a function and display the information in that function.

```
#include <stdio.h>
struct student{
    char name[50];
    int roll;
};
void Display(struct student stu);
/* function prototype should be below to the structure declaration otherwise
compiler shows error */
int main(){
    struct student s1;
    printf("Enter student's name: ");
    scanf("%s",&s1.name);
    printf("Enter roll number:");
    scanf("%d",&s1.roll);
    Display(s1);   // passing structure variable s1 as argument
    return 0;
}
void Display(struct student stu){
  printf("Output\nName: %s",stu.name);
  printf("\nRoll: %d",stu.roll);
}
```

Output

```
Enter student's name: Akshay jadhv
Enter roll number: 149
Output
Name: Akshay jadhv
Roll: 149
```

Unions:

Unions are quite similar to the structures in C. Union is also a derived type as structure. Union can be defined in same manner as structures just the keyword used in defining union in **union** where keyword used in defining structure was **strut.**

Syntax:

```
union car{

  char name[50];

  int price;};
```

Advantage of union over structure

It occupies less memory because it occupies the memory of largest member only.

Disadvantage of union over structure

It can store data in one member only.

Difference between union and structure

Though unions are similar to structure in so many ways, the difference between them is crucial to understand. This can be demonstrated by this example:

Defining a union

Defining a Structure

```
#include <stdio.h>
union job {
   char name[32];
   float salary;
   int worker_no;
}u;
struct job1 {
   char name[32];
   float salary;
   int worker_no;
}s;
int main(){
   printf("size of union = %d",sizeof(u));
   printf("\nsize of structure = %d", sizeof(s));
   return 0;
}
```

Output

```
size of union = 32

size of structure = 40
```

There is difference in memory allocation between union and structure as suggested in above example. The amount of memory required to store a structure variables is the sum of memory size of all members.

Difference between Structure and Union

	Structure	Union
1	For defining structure use struct keyword.	For defining union we use union keyword
2	Structure occupies more memory space than union.	Union occupies less memory space than Structure.
3	In Structure we can access all members of structure at a time.	In union we can access only one member of union at a time.
4	Structure allocates separate storage space for its every members.	Union allocates one common storage space for its all members. Union find which member need more memory than other member, then it allocate that much space

Union Example:

As you know, all members of structure can be accessed at any time. But, only one member of union can be accessed at a time in case of union and other members will contain garbage value.

```
#include <stdio.h>
union job {
  char name[32];
  float salary;
  int worker_no;
}u;
int main(){
  printf("Enter name:\n");
  scanf("%s",&u.name);
  printf("Enter salary: \n");
  scanf("%f",&u.salary);
  printf("Displaying\nName :%s\n",u.name);
  printf("Salary: %f",u.salary);
  return 0;
}
```

Output

```
Enter name
smita
Enter salary
99.66
Displaying
Name: f%Bary
Salary: 99.66
```

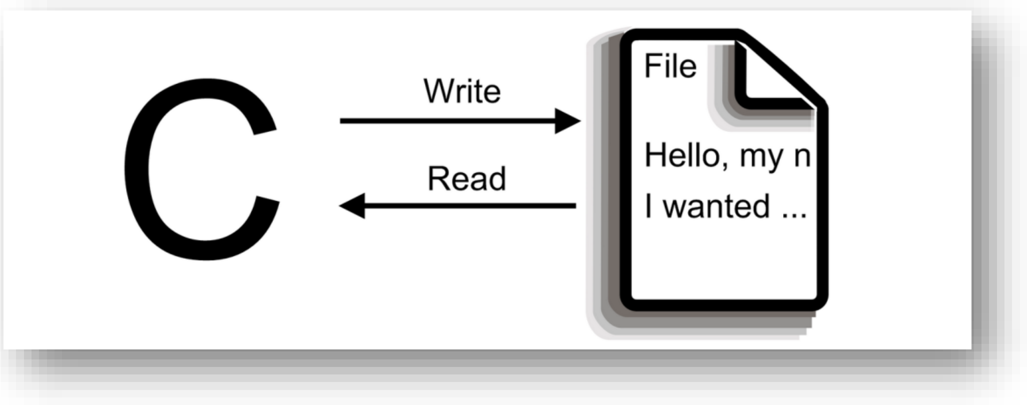
Chapter 8-Exercise

1	Write a program in C to add two numbers using pointers
2	Write a program in C to swap elements using call by reference
3	Write a C program to read and print elements of array.
4	Write a C program to print all negative elements in an array.
5	Write a C program to find sum of all array elements.
6	Write a C program to find maximum and minimum element in an array.
7	Write a C program to add two matrices.
8	Write a C program to subtract two matrices.
9	Write a C program to multiply two matrices.

Chapter 9:File handling

File Handling:

File Handling concept in C language is used for store a data permanently in computer. Using this concept we can store our data in Secondary memory (Hard disk). All files related function are available in **stdio.h** header file.



How to achieve File Handling in C

For achieving file handling in C we need follow following steps

- Naming a file
- Opening a file
- Reading data from file
- Writing data into file
- Closing a file.

Why files are needed?

When the program is terminated, the entire data is lost in C programming. If you want to keep large volume of data, it is time consuming to enter the entire data. But, if file is created, these information can be accessed using few commands. There are large numbers of functions to handle file I/O in C language.

Advantage of File:

It will contain the data even after program exit. Normally we use variable or array to store data, but data is lost after program exit. Variables and arrays are non-permanent storage medium whereas file is permanent storage medium.

Working with file

While working with file, you need to declare a pointer of type file. This declaration is needed for communication between file and program

Syntax:

```
FILE *ptr;
```

Functions use in File Handling in C

S.No	Function	Operation
1	fopen()	To create a file
2	fclose()	To close an existing file

3	getc()	Read a character from a file
4	putc()	Write a character in file
5	fprintf()	To write set of data in file
6	fscanf()	To read set of data from file.

File Opening Mode:

S.No	Mode	Meaning	Purpose
1	R	Reading	Open the file for reading only.
2	W	Writing	Open the file for writing only.
3	A	Appending	Open the file for appending (or adding) data to it.

Writing to a file:

Opening a file

Printing the Text in to File

```
#include <stdio.h>
int main()
{
  int n;
  FILE *fptr;
  fptr=fopen("C:\\program.txt","w");
  if(fptr==NULL){
    printf("Error!");
    exit(1);
  }
  printf("Enter n: ");
  scanf("%d",&n);
  fprintf(fptr,"%d",n);
  fclose(fptr);
  return 0;
}
```

This program takes the number from user and stores in file. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open that file, you can see the integer you entered.

Similarly, **fscanf()** can be used to read data from file.

Reading from file

Reading the text from file

```
#include <stdio.h>
int main()
{
  int n;
  FILE *fptr;
  if ((fptr=fopen("C:\\program.txt","r"))==NULL){
     printf("Error! opening file");
     exit(1);       /* Program exits if file pointer returns NULL. */
  }
  fscanf(fptr,"%d",&n);
  printf("Value of n=%d",n);
  fclose(fptr);
  return 0;
}
```

C Programming Enumeration:

An enumeration is a user-defined data type consists of integral constants and each integral constant is give a name. Keyword **enum** is used to defined enumerated data type.

```
enum type_name{ value1, value2,...,valueN };
```

Here, **type_name** is the name of enumerated data type or tag. And **value1**, **value2**,....,**valueN**are values of type **type_name**.

By default, **value1** will be equal to 0, **value2** will be 1 and so on but, the programmer can change the default value.

```
// Changing the default value of enum elements
enum suit{
    club=0;
    diamonds=10;
    hearts=20;
    spades=3;
};
```

Declaration of enumerated variable

Above code defines the type of the data but, no any variable is created. Variable of type **enum**can be created as:

```
enum boolean{
    false;
    true;
};
enum boolean check;
```

Here, a variable check is declared which is of type **enum boolean**.

Example of enumerated type

```
#include <stdio.h>
enum week{ sunday, monday, tuesday, wednesday, thursday, friday,
saturday};
int main(){
    enum week today;
    today=wednesday;
    printf("%d day",today+1);
    return 0;
   }
```

Output

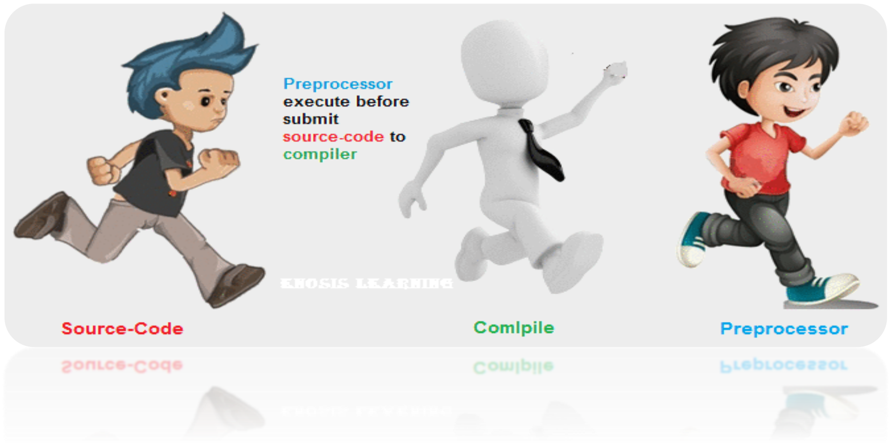
```
            4day
```

No	Programs
1	Write a C Program to copy the contents of one file into another using fputc()
2	Write a C Program to read last n characters from the file !
3	Write a C Program to convert the file contents in Upper-case & Write Contents in a output file
4	Write a C Program to Compare two text/data files in C Programming
5	Write a C Program to Write content on Data File and Read From Data File
6	Write a C Program to Copy Text From One File to Other File

Chapter 10: Pre-processor Directives

pre-processor :

Preprocessor is a program which will executed automatically before passing the source program to compiler. This process is called pre-processing. The preprocessor provides the ability for the inclusion of header files, macro expansions, conditional compilation, and line control.



Defined Preprocessor Directive:

Preprocessor Directive can be place any where in the program, but generally it place top of the program before defining the first function.

Use of #include

Let us consider very common preprocessing directive as below:

#include <stdio.h>

Here, **"stdio.h"** is a header file and the preprocessor replace the above line with the contents of header file.

Use of #define

Preprocessing directive #define has two forms.

The first form is:

```
#define identifier token_string
```

token_string part is optional but, are used almost every time in program.

C Program to find are

a of a cricle. [Area of circle=πr^{2}]

```
#include <stdio.h>
#define PI 3.1415
int main(){
    int radius;
    float area;
    printf("Enter the radius: ");
    scanf("%d",&radius);
    area=PI*radius*radius;
    printf("Area=%.2f",area);
    return 0;
}
```

Output

```
Enter the radius: 3
Area=28.27
```

Macros:

A macro is a segment of code which is replaced by the value of macro. Macro is defined by #define directive.

Predefined Macros in C language

Predefined macro	Value
__DATE__	String containing the current date
__FILE__	String containing the file name
__LINE__	Integer representing the current line number
__STDC__	If follows ANSI standard C, then value is a nonzero integer
__TIME__	String containing the current date.

How to use predefined Macros?

C Program to find the current time

```
#include <stdio.h>

int main(){

   printf("Current time: %s",__TIME__);   //calculate the current
time

}
```

Output:

```
                    Current time: 19:54:39
```