**Structure of C Language program**

1 ) Comment line

2) Preprocessor directive

3 ) Global variable declaration

4) main function( )

{

Local variables;

Statements;

}

User defined function

}

}

Comment line It indicates the purpose of the program. It is represented as /\*……………………………..\*/ Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed within the decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & character constant.

Preprocessor Directive:

#include<stdio.h> tells the compiler to include information about the standard input/output library. It is also used in symbolic constant such as #define PI 3.14(value). The stdio.h (standard input output header file) contains definition &declaration of system defined function such as printf( ), scanf( ), pow( ) etc.  
Generally printf() function used to display and scanf() function used to read value  
**Global Declaration:**This is the section where variable are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function :

main()It is the user defined function and every function has one main() function from where actually program is started and it is encloses within the pair of curly braces. The main( ) function can be anywhere in the program but in general practice it is placed in the first position.  
Syntax :  
main()

{  
……..  
……..  
……..  
}  
The main( ) function return value when it declared by data type as  
int main( )  
{  
return 0

}  
The main function does not return any value when void (means null/empty) as  
void main(void ) or void main()  
{  
printf (“C language”);  
}  
Output: C language

Output: C language  
The program execution start with opening braces and end with closing brace.  
And in between the two braces declaration part as well as executable part is  
mentioned. And at the end of each line, the semi-colon is given which indicates  
statement termination.

int main (void)  
{  
printf ("welcome to c Programming language.\n");  
return 0;  
}  
Output: welcome to c programming language.  
**Steps for Compiling and executing the Programs**

A compiler is a software program that analyzes a program developed in a particular  
computer language and then translates it into a form that is suitable for execution

on a particular computer system. Figure below shows the steps that are involved in  
entering, compiling, and executing a  
computer program developed in the C programming language and the typical Unix  
commands that would be entered from the command line.

/\* Simple program to add two numbers…………………….\*/

#include <stdio.h>  
int main (void)  
{  
int v1, v2, sum; //v1,v2,sum are variables and int is data type declared  
v1 = 150;  
v2 = 25;  
sum = v1 + v2;  
printf ("The sum of %i and %i is= %i\n", v1, v2, sum);  
return 0;  
}  
Output:  
The sum of 150 and 25 is=175  
**Character set**A character denotes any alphabet, digit or special symbol used to represent information. Valid alphabets, numbers and special symbols allowed in C are The alphabets, numbers and special symbols when properly combined form constants, variables and keywords.  
**Identifiers**Identifiers are user defined word used to name of entities like variables, arrays,  
functions, structures etc. Rules for naming identifiers are:  
1) name should only consists of alphabets (both upper and lower case), digits  
and underscore (\_) sign.  
2) first characters should be alphabet or underscore  
3) name should not be a keyword  
4) since C is a case sensitive, the upper case and lower case considered differently, for example code, Code, CODE etc. are different identifiers.  
5) identifiers are generally given in some meaningful name such as value, net\_salary, age, data etc. An identifier name may be long, some implementation recognizes only first eight characters, most recognize 31 characters. ANSI  
standard compiler recognize 31 characters. Some invalid identifiers are 5cb, int, res#, avg no etc.  
**Keyword**

There are certain words reserved for doing specific task, these words are known as reserved word or keywords. These words are predefined and always written in lower case or small letter. These keywords cann’t be used as a variable name as it assigned with fixed meaning. Some examples are int, short, signed, unsigned, default, volatile, float, long, double, break, continue, typedef, static, do, for, union, return, while, do, extern, register, enum, case, goto, struct, char, auto, const etc.  
**data types**  
Data types refer to an extensive system used for declaring variables or functions of different types before its use. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. The value of a variable can be changed any time.  
C has the following 4 types of data types  
**basic built-in data types**: int, float, double, char  
**Enumeration data type**: enum  
**Derived data type**: pointer, array, structure, union  
**Void data type**: void  
A variable declared to be of type int can be used to contain integral values only—that is, values that do not contain decimal places. A variable declared to be of type float can be used for storing floating- point numbers (values containing decimal places). The double type is the same as type float, only with roughly twice the precision. The char data type can be used to store a single character, such as the  
letter a, the digit character 6, or a semicolon similarly A variable declared char can only store character type value.  
There are two types of type qualifier in c  
Size qualifier: short, long  
Sign qualifier: signed, unsigned

When the qualifier unsigned is used the number is always positive, and when signed is used number may be positive or negative. If the sign qualifier is not mentioned, then by default sign qualifier is assumed. The range of values for signed data types is less than that of unsigned data type. Because in signed type, the left most bit is used to represent sign, while in unsigned type this bit is also used to represent the value. The size and range of the different data types on a 16 bit machine is given below:

|  |  |  |  |
| --- | --- | --- | --- |
| Basic data type | Data type with type qualifier | Size (byte) | Range |
| char | char or signed char Unsigned char | 11 | -128 to 127 0 to 255 |
| int | int or signed int unsigned int short int or signed short int unsigned short int long int or signed long int unsigned long int | 221144 | -32768 to 32767 0 to 65535 -128 to 127 0 to 255 -2147483648 to 2147483647 0 to 4294967295 |
| float | float | 4 | -3.4E-38 to 3.4E+38 |
| double | double Long double | 8 10 | 1.7E-308 to 1.7E+308 3.4E-4932 to 1.1E+493 |

ConstantsConstant is a any value that cannot be changed during program execution. In C, any number, single character, or character string is known as a *constant*. A constant is an entity that doesn’t change whereas a variable is an entity that may change.  
For example, the number 50 represents a constant integer value. The character string "Programming in C is fun.\n" is an example of a constant character string. C constants can be divided into two major categories:  
**Primary Constants  
Secondary Constants**  
These constants are further categorized as

**Numeric constant  
Character constant  
String constant**

**Numeric constant:** Numeric constant consists of digits. It required minimum size of 2 bytes and max 4 bytes. It may be positive or negative but by default sign is always positive. No comma or space is allowed within the numeric constant and it must have at least 1 digit. The allowable range for integer constants is -32768 to 32767. Truly speaking the range of an Integer constant depends upon the compiler.  
For a 16-bit compiler like Turbo C or Turbo C++ the range is –32768 to 32767.  
For a 32-bit compiler the range would be even greater. Mean by a 16-bit or a 32- bit compiler, what range of an Integer constant has to do with the type of compiler. It is categorized a integer constant and real constant. An integer constants are whole number which have no decimal point. Types of integer constants are:  
**Decimal constant: 0-------9(base 10)  
Octal constant: 0-------7(base 8)  
Hexa decimal constant: 0----9, A------F(base 16)**  
In decimal constant first digit should not be zero unlike octal constant first digit must be zero(as 076, 0127) and in hexadecimal constant first two digit should be 0x/ 0X (such as 0x24, 0x87A). By default type of integer constant is integer but if the value of integer constant is exceeds range then value represented by integer type is taken to be unsigned integer or long integer. It can also be explicitly  
mention integer and unsigned integer type by suffix l/L and u/U.  
**Real constant** is also called floating point constant. To construct real constant we  
must follow the rule of ,  
-real constant must have at least one digit.  
-It must have a decimal point.  
-It could be either positive or negative.  
-Default sign is positive.  
-No commas or blanks are allowed within a real constant. Ex.: +325.34  
426.0  
-32.76  
To express small/large real constant exponent(scientific) form is used where number is written in mantissa and exponent form separated by e/E. Exponent can be positive or negative integer but mantissa can be real/integer type, for example 3.6\*105=3.6e+5. By default type of floating point constant is double, it can also be explicitly defined it by suffix of f/F.  
**Character constant**

Character constant represented as a single character enclosed within a single quote. These can be single digit, single special symbol or white spaces such as ‘9’,’c’,’$’, ‘ ’ etc. Every character constant has a unique integer like value in machine’s character code as if machine using ASCII (American standard code for information interchange). Some numeric value associated with each upper and  
lower case alphabets and decimal integers are as:  
A------------ Z ASCII value (65-90)  
a-------------z ASCII value (97-122)  
0-------------9 ASCII value (48-59)  
; ASCII value (59)  
**String constant**Set of characters are called string and when sequence of characters are enclosed within a double quote (it may be combination of all kind of symbols) is a string constant. String constant has zero, one or more than one character and at the end of the string null character(\0) is automatically placed by compiler. Some examples are “,sarathina” , “908”, “3”,” ”, “A” etc. In C although same characters are enclosed within single and double quotes it represents different meaning such as “A” and ‘A’ are different because first one is string attached with null character at the end but second one is character constant with its corresponding ASCII value is 65.  
**Symbolic constant**

**Symbolic constant**Symbolic constant is a name that substitute for a sequence of characters and,  
characters may be numeric, character or string constant. These constant are  
generally defined at the beginning of the program as  
#define name value , here name generally written in  
upper case for example

#define MAX 10  
#define CH ‘b’  
#define NAME “sony”  
Variables  
Variable is a data name which is used to store some data value or symbolic names  
for storing program  
computations and results. The value of the variable can be change during the  
execution. The rule for naming the variables is same as the naming identifier. Before used in the program it must be declared. Declaration of variables specify its name, data types and range of the value that variables can store depends upon its data types.  
Syntax:  
int a;  
char c;  
float f;  
Variable initialization  
When we assign any initial value to variable during the declaration, is called initialization of variables. When variable is declared but contain undefined value then it is called garbage value. The variable is initialized with the assignment operator such as Data type variable name=constant;  
Example: int a=20;  
Or int a;  
a=20;

ExpressionsAn expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical and relational for example:-  
int z= x+y // arithmatic expression  
a>b //relational  
a==b // logical  
func(a, b) // function call  
Expressions consisting entirely of constant values are called *constant expressions*. So, the expression  
121 + 17 – 110 is a constant expression because each of the terms of the expression is a constant  
value. But if i were declared to be an integer variable, the expression 180 + 2 – j would not represent a constant expression.  
**Operator**  
This is a symbol use to perform some operation on variables, operands or with the constant. Some operator required 2 operand to perform operation or Some required single operation. Several operators are there those are, arithmetic operator, assignment, increment , decrement, logical, conditional, comma, size of , bitwise and others.  
**1. Arithmatic Operator**  
This operator used for numeric calculation. These are of either Unary arithmetic operator, Binary arithmetic operator. Where Unary arithmetic operator required only one operand such as +,-, ++, --,!, tiled. And these operators are addition, subtraction, multiplication, division. Binary arithmetic operator on other hand required two operand and its operators are +(addition), -(subtraction), \*(multiplication), /(division), %(modulus). But modulus cannot applied with floating point operand as well as there are no exponent operator in c. Unary (+) and Unary (-) is different from addition and subtraction. When both the operand are integer then it is called integer arithmetic and the result is always integer. When both the operand are floating point then it is called floating arithmetic and when operand is of integer and floating point then it is called mix type or mixed mode arithmetic . And the result is in float type.  
**2.Assignment Operator**  
A value can be stored in a variable with the use of assignment operator. The assignment operator(=) is used in assignment statement and assignment expression. Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression. When variable on the left hand side is occur on the right hand side then we can avoid by writing the  
compound statement. For example,  
int x= y;  
int Sum=x+y+z;  
3.Increment and Decrement  
The Unary operator ++, --, is used as increment and decrement which acts upon single operand. Increment operator increases the value of variable by one .Similarly decrement operator decrease the value of the variable by one. And these operator can only used with the variable, but cann't use with expression and constant as ++6 or ++(x+y+z).

It again categories into prefix post fix . In the prefix the value of the variable is incremented 1st, then the new value is used, where as in postfix the operator is written after the operand(such as m++,m--).  
EXAMPLE  
let y=12;  
z= ++y;  
y= y+1;  
z= y;  
Similarly in the postfix increment and decrement operator is used in the operation .  
And then increment and decrement is perform.  
EXAMPLE  
let x= 5;  
y= x++;  
y=x;  
x= x+1;  
**4.Relational Operator**  
It is use to compared value of two expressions depending on their relation. Expression that contain relational operator is called relational expression. Here the value is assign according to true or false value.  
a.(a>=b) || (b>20)  
b.(b>a) && (e>b)  
c. 0(b!=7)  
**5. Conditional Operator**

It sometimes called as ternary operator. Since it required three expressions as  
operand and it is represented as (? , :).  
SYNTAX  
exp1 ? exp2 :exp3  
Here exp1 is first evaluated. It is true then value return will be exp2 . If false then  
exp3.  
EXAMPLE  
void main()  
{  
int a=10, b=2  
int s= (a>b) ? a:b;  
printf(“value is:%d”);  
}  
Output:  
Value is:10  
6. Comma Operator  
Comma operator is use to permit different expression to be appear in a situation  
where only one expression would be used. All the expression are separator by  
comma and are evaluated from left to right.  
EXAMPLE  
int i, j, k, l;  
for(i=1,j=2;i<=5;j<=10;i++;j++)

**Control Statement**  
Generally C program statement is executed in a order in which they appear in the program. But sometimes we use decision making condition for execution nly a part of program, that is called control statement. Control statement defined how the control is transferred from one part to the other part of the program. There are several control statement like if...else, switch, while, do....while, for loop,  
break, continue, goto etc.  
**Loops in C**Loop:-it is a block of statement that performs set of instructions. In loops  
Repeating particular portion of the program either a specified number of time or  
until a particular no of condition is being satisfied.  
There are three types of loops in c  
**1.While loop  
2.do while loop  
3.for loop  
While loop**Syntax:-  
while(condition)  
{  
Statement 1;  
Statement 2;  
}  
Or while(test condition)  
Statement;

The test condition may be any expression .when we want to do something a fixed  
no of times but not known about the number of iteration, in a program then while  
loop is used.  
Here first condition is checked if, it is true body of the loop is executed else, If  
condition is false control will be come out of loop.  
Example:-  
/\* wap to print 5 times welcome to C” \*/  
#include<stdio.h>  
void main()  
{  
int p=1;  
While(p<=5)  
{  
printf(“Welcome to C\n”);  
P=p+1;  
} }  
Output: Welcome to C  
Welcome to C  
Welcome to C  
Welcome to C  
Welcome to C

So as long as condition remains true statements within the body of while loop will  
get executed repeatedly.  
**do while loop**This (do while loop) statement is also used for looping. The body of this loop may  
contain single statement or block of statement. The syntax for writing this  
statement is:  
Syntax:-  
Do  
{  
Statement;  
}  
while(condition);  
Example:-  
#include<stdio.h>  
void main()  
{  
int X=4;  
do  
{  
Printf(“%d”,X);  
X=X+1;

}whie(X<=10);  
Printf(“ ”);  
}  
Output: 4 5 6 7 8 9 10  
Here firstly statement inside body is executed then condition is checked. If the  
condition is true again body of loop is executed and this process continue until the  
condition becomes false. Unlike while loop semicolon is placed at the end of  
while.  
There is minor difference between while and do while loop, while loop test the  
condition before executing any of the statement of loop. Whereas do while loop  
test condition after having executed the statement at least one within the loop.  
If initial condition is false while loop would not executed it’s statement on other  
hand do while loop executed it’s statement at least once even If condition fails for  
first time. It means do while loop always executes at least once.

**Notes:**Do while loop used rarely when we want to execute a loop at least once.

**for loop**In a program, for loop is generally used when number of iteration are known in  
advance. The body of the loop can be single statement or multiple statements. Its  
syntax for writing is:  
Syntax:-

for(exp1;exp2;exp3)  
{  
Statement;  
}  
Or  
for(initialized counter; test counter; update counter)  
{  
Statement;  
}

Here exp1 is an initialization expression, exp2 is test expression or condition and  
exp3 is an update expression. Expression 1 is executed only once when loop  
started and used to initialize the loop variables. Condition expression generally  
uses relational and logical operators. And updation part executed only when after  
body of the loop is executed.  
Example:-  
void main()  
{

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

Printf(“ %d ”, i);  
}  
}  
Output:-1 2 3 4 5 6 7 8 9  
**Nesting of loop**When a loop written inside the body of another loop then, it is known as nesting of  
loop. Any type of loop can be nested in any type such as while, do while, for. For  
example nesting of for loop can be represented as :  
void main()  
{  
int i,j;  
for(i=0;i<2;i++)  
for(j=0;j<5; j++)  
printf(“%d %d”, i, j);  
}  
Output: i=0  
j=0 1 2 3 4  
i=1  
j=0 1 2 3 4

**Break statement(break)**Sometimes it becomes necessary to come out of the loop even before loop  
condition becomes false then break statement is used. Break statement is used  
inside loop and switch statements. It cause immediate exit from that loop in which  
it appears and it is generally written with condition. It is written with the keyword  
as **break.** When break statement is encountered loop is terminated and control is  
transferred to the statement, immediately after loop or situation where we want to  
jump out of the loop instantly without waiting to get back to conditional state.  
When break is encountered inside any loop, control automatically passes to the  
first statement after the loop. This break statement is usually associated with **if**statement.

Example :  
void main()  
{  
int j=0;  
for(;j<6;j++)  
if(j==4)  
break;  
}  
Output:  
0 1 2 3  
**Continue statement (key word continue)**

Continue statement is used for continuing next iteration of loop after skipping  
some statement of loop. When it encountered control automatically passes  
through the beginning of the loop. It is usually associated with the if statement. It is  
useful when we want to continue the program without executing any part of the  
program.  
The difference between break and continue is, when the break encountered loop is  
terminated and it transfer to the next statement and when continue is encounter  
control come back to the beginning position.  
In while and do while loop after continue statement control transfer to the test  
condition and then loop continue where as in, for loop after continue control  
transferred to the updating expression and condition is tested.  
Example:-  
void main()  
{  
int n;  
for(n=2; n<=9; n++)  
{  
if(n==4)  
continue;  
printf(“%d”, n);  
}  
}  
Printf(“out of loop”);  
}  
Output: 2 3 5 6 7 8 9 out of loop

**if statement**Statement execute set of command like when condition is true and its syntax is  
If (condition)  
Statement;  
The statement is executed only when condition is true. If the if statement body is  
consists of several statement then better to use pair of curly braces. Here in case  
condition is false then compiler skip the line within the if block.  
void main()  
{  
int n;  
printf (“ enter a number:”);  
scanf(“%d”,&n);  
If (n>10)  
Printf(“ number is grater”);  
}  
Output:  
Enter a number:12  
Number is greater

**if…..else ... Statement**it is bidirectional conditional control statement that contains one condition & two  
possible action. Condition may be true or false, where non-zero value regarded as  
true & zero value regarded as false. If condition are satisfy true, then a single or  
block of statement executed otherwise another single or block of statement is  
executed.  
Its syntax is:-  
if (condition)  
{  
Statement1;  
Statement2;  
}  
else  
{  
Statement1;  
Statement2;  
}  
Else statement cannot be used without if or no multiple else statement are allowed  
within one if statement. It means there must be a if statement with in an else  
statement.  
Example:-  
/\* To check a number is eve or odd \*/

void main()  
{  
int n;  
printf (“enter a number:”);  
sacnf (“%d”, &n);  
If (n%2==0)  
printf (“even number”);  
else  
printf(“odd number”);  
}  
Output: enter a number:121  
odd number

**Nesting of if …else**When there are another if else statement in if-block or else-block, then it is called  
nesting of if-else statement.  
Syntax is :-  
if (condition)  
{

If (condition)  
Statement1;  
else  
statement2;  
}  
Statement3;  
**If….else LADDER**In this type of nesting there is an if else statement in every else part except the last  
part. If condition is false control pass to block where condition is again checked  
with its if statement.  
Syntax is :-  
if (condition)  
Statement1;  
else if (condition)  
statement2;  
else if (condition)  
statement3;  
else  
statement4;  
This process continue until there is no if statement in the last block. if one of the  
condition is satisfy the condition other nested “else if” would not executed.

But it has disadvantage over if else statement that, in if else statement whenever  
the condition is true, other condition are not checked. While in this case, all  
condition are checked.

ARRAYArray is the collection of similar data types or collection of similar entity stored in  
contiguous memory location. Array of character is a string. Each data item of an  
array is called an element. And each element is unique and located in separated  
memory location. Each of elements of an array share a variable but each element  
having different index no. known as subscript.  
An array can be a single dimensional or multi-dimensional and number of  
subscripts determines its dimension. And number of subscript is always starts with  
zero. One dimensional array is known as vector and two dimensional arrays are  
known as matrix.  
ADVANTAGES: array variable can store more than one value at a time where  
other variable can store one value at a time.  
Example:  
int arr[100];

int mark[100];  
**DECLARATION OF AN ARRAY :**Its syntax is :  
Data type array name [size];  
int arr[100];  
int mark[100];  
int a[5]={10,20,30,100,5}  
The declaration of an array tells the compiler that, the data type, name of the array,  
size of the array and for each element it occupies memory space. Like for int data  
type, it occupies 2 bytes for each element and for float it occupies 4 byte for each  
element etc. The size of the array operates the number of elements that can be  
stored in an array and it may be a int constant or constant int expression.  
We can represent individual array as :  
int ar[5];  
ar[0], ar[1], ar[2], ar[3], ar[4];  
Symbolic constant can also be used to specify the size of the array as:  
#define SIZE 10;  
**INITIALIZATION OF AN ARRAY:**After declaration element of local array has garbage value. If it is global or static  
array then it will be automatically initialize with zero. An explicitly it can be  
initialize that  
Data type array name [size] = {value1, value2, value3…}  
Example:  
in ar[5]={20,60,90, 100,120}

Array subscript always start from zero which is known as lower bound and upper  
value is known as upper bound and the last subscript value is one less than the size  
of array. Subscript can be an expression i.e. integer value. It can be any integer,  
integer constant, integer variable, integer expression or return value from  
functional call that yield integer value.  
So if i & j are not variable then the valid subscript are  
ar [i\*7],ar[i\*i],ar[i++],ar[3];  
The array elements are standing in continuous memory locations and the  
amount of storage required for hold the element depend in its size & type.  
**Total size in byte for 1D array is:**Total bytes=size of (data type) \* size of array.  
Example : if an array declared is:  
int [20];  
Total byte= 2 \* 20 =40 byte.  
**ACCESSING OF ARRAY ELEMENT:**/\*Write a program to input values into an array and display them\*/  
#include<stdio.h>  
int main()  
{  
int arr[5],i;  
for(i=0;i<5;i++)  
{  
printf(“enter a value for arr[%d] \n”,i);  
scanf(“%d”,&arr[i]);  
}

printf(“the array elements are: \n”);  
for (i=0;i<5;i++)  
{  
printf(“%d\t”,arr[i]);  
}  
return 0;  
}  
OUTPUT:  
Enter a value for arr[0] = 12  
Enter a value for arr[1] =45  
Enter a value for arr[2] =59  
Enter a value for arr[3] =98  
Enter a value for arr[4] =21  
The array elements are 12 45 59 98 21  
Example: From the above example value stored in an array are and occupy its  
memory addresses 2000, 2002, 2004, 2006, 2008 respectively.  
a[0]=12, a[1]=45, a[2]=59, a[3]=98, a[4]=21  
ar[0] ar[1] ar[2] ar[3] ar[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 45 | 59 | 98 | 21 |

2000 2002 2004 2006 2008  
Example 2:

/\* Write a program to add 10 array elements \*/  
#include<stdio.h>  
void main()  
{  
int i ;  
int arr [10];  
int sum=o;  
for (i=0; i<=9; i++)  
{  
printf (“enter the %d element \n”, i+1);  
scanf (“%d”, &arr[i]);  
}  
for (i=0; i<=9; i++)  
{  
sum = sum + a[i];  
}  
printf (“the sum of 10 array elements is %d”, sum);  
}  
OUTPUT:  
Enter a value for arr[0] =5  
Enter a value for arr[1] =10  
Enter a value for arr[2] =15  
Enter a value for arr[3] =20

Enter a value for arr[4] =25  
Enter a value for arr[5] =30  
Enter a value for arr[6] =35  
Enter a value for arr[7] =40  
Enter a value for arr[8] =45  
Enter a value for arr[9] =50  
Sum = 275  
while initializing a single dimensional array, it is optional to specify the size of  
array. If the size is omitted during initialization then the compiler assumes the size  
of array equal to the number of initializers.  
For example:-  
int marks[]={99,78,50,45,67,89};  
If during the initialization of the number the initializers is less then size of array,  
then all the remaining elements of array are assigned value zero .  
For example:-  
int marks[5]={99,78};  
Here the size of the array is 5 while there are only two initializers so After this  
initialization, the value of the rest elements are automatically occupied by zeros  
such as  
Marks[0]=99 , Marks[1]=78 , Marks[2]=0, Marks[3]=0, Marks[4]=0  
Again if we initialize an array like  
int array[100]={0};  
Then the all the element of the array will be initialized to zero. If the number of  
initializers is more than the size given in brackets then the compiler will show an  
error.

**Single dimensional arrays and functions**/\*program to pass array elements to a function\*/  
#include<stdio.h>  
void main()  
{  
int arr[10],i;  
printf(“enter the array elements\n”);  
for(i=0;i<10;i++)  
{  
scanf(“%d”,&arr[i]);  
check(arr[i]);  
} }

void check(int num)  
{  
if(num%2=0)  
{  
printf(”%d is even \n”,num);  
}  
else  
{  
printf(”%d is odd \n”,num);  
}  
}

**Two dimensional arrays**Two dimensional array is known as matrix. The array declaration in both the array  
i.e.in single dimensional array single subscript is used and in two dimensional  
array two subscripts are is used.  
Its syntax is  
Data-type array name[row][column];  
Or we can say 2-d array is a collection of 1-D array placed one below the other.

Total no. of elements in 2-D array is calculated as **row\*column**Example:-  
int a[2][3];  
Total no of elements=row\*column is 2\*3 =6  
It means the matrix consist of 2 rows and 3 columns  
For example:-  
20 2 7  
8 3 15  
Positions of 2-D array elements in an array are as below  
00 01 02  
10 11 12  
a [0][0] a [0][0] a [0][0] a [0][0] a [0][0] a [0][0]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 20 | 2 | 7 | 8 | 3 | 15 |

2000 2002 2004 2006 2008  
**Accessing 2-d array /processing 2-d arrays**For processing 2-d array, we use two nested for loops. The outer for loop  
corresponds to the row and the inner for loop corresponds to the column.  
For example  
int a[4][5];  
**for reading value:-**

for(i=0;i<4;i++)  
{  
for(j=0;j<5;j++)  
{  
scanf(“%d”,&a[i][j]);  
}  
}  
For displaying value:-  
for(i=0;i<4;i++)  
{  
for(j=0;j<5;j++)  
{  
printf(“%d”,a[i][j]);  
}  
}  
**Initialization of 2-d array:**2-D array can be initialized in a way similar to that of 1-D array. for example:-  
int mat[4][3]={11,12,13,14,15,16,17,18,19,20,21,22};  
These values are assigned to the elements row wise, so the values of  
elements after this initialization are  
Mat[0][0]=11, Mat[1][0]=14, Mat[2][0]=17 Mat[3][0]=20  
Mat[0][1]=12, Mat[1][1]=15, Mat[2][1]=18 Mat[3][1]=21  
Mat[0][2]=13, Mat[1][2]=16, Mat[2][2]=19 Mat[3][2]=22

While initializing we can group the elements row wise using inner braces.  
for example:-  
int mat[4][3]={{11,12,13},{14,15,16},{17,18,19},{20,21,22}};  
And while initializing , it is necessary to mention the 2nd dimension where 1st  
dimension is optional.  
int mat[][3];  
int mat[2][3];  
int mat[][];  
int mat[2][]; invalid  
If we **initialize an array** as  
int mat[4][3]={{11},{12,13},{14,15,16},{17}};  
Then the compiler will assume its all rest value as 0,which are not defined.  
Mat[0][0]=11, Mat[1][0]=12, Mat[2][0]=14, Mat[3][0]=17  
Mat[0][1]=0, Mat[1][1]=13, Mat[2][1]=15 Mat[3][1]=0  
Mat[0][2]=0, Mat[1][2]=0, Mat[2][2]=16, Mat[3][2]=0  
In memory map whether it is 1-D or 2-D, elements are stored in one  
contiguous manner.  
We can also give the size of the 2-D array by using symbolic constant  
Such as  
#define ROW 2;

#define COLUMN 3;  
int mat[ROW][COLUMN];  
**String**Array of character is called a string. It is always terminated by the NULL  
character. String is a one dimensional array of character.  
We can initialize the string as  
char name[]={‘j’,’o’,’h’,’n’,’\o’};  
Here each character occupies 1 byte of memory and last character is always NULL  
character. Where ’\o’ and 0 (zero) are not same, where **ASCII** value of ‘\o’ is 0  
and ASCII value of 0 is 48. Array elements of character array are also stored in  
contiguous memory allocation.  
From the above we can represent as;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| J | o | h | N | ‘\o’ |

The terminating NULL is important because it is only the way that the  
function that work with string can know, where string end.  
String can also be **initialized** as;  
char name[]=”John”;  
Here the NULL character is not necessary and the compiler will assume it  
automatically.  
**String constant (string literal)** A string constant is a set of character that enclosed within the double quotes  
and is also called a literal. Whenever a string constant is written anywhere in a  
program it is stored somewhere in a memory as an array of characters terminated  
by a NULL character (‘\o’).  
Example – “m”  
“Tajmahal”  
“My age is %d and height is %f\n”  
The string constant itself becomes a pointer to the first character in array.  
Example-char crr[20]=”Taj mahal”;  
It is called base address

**String library function**There are several string library functions used to manipulate string and the  
prototypes for these functions are in header file “string.h”. Several string functions  
are  
**strlen()**This function return the length of the string. i.e. the number of characters in the  
string excluding the terminating NULL character.  
It accepts a single argument which is pointer to the first character of the string.

For examplestrlen(“suresh”);  
It return the value 6.  
**In array version to calculate legnth:**-  
int str(char str[])  
{  
int i=0;  
while(str[i]!=’\o’)  
{  
i++;  
}  
return i;  
}  
Example:-  
#include<stdio.h>  
#include<string.h>  
void main()  
{  
char str[50];  
print(”Enter a string:”);

gets(str);  
printf(“Length of the string is %d\n”,strlen(str));  
}  
Output:  
Enter a string: C in Depth  
Length of the string is 8  
**strcmp()**This function is used to compare two strings. If the two string match, strcmp()  
return a value 0 otherwise it return a non-zero value. It compare the strings  
character by character and the comparison stops when the end of the string is  
reached or the corresponding characters in the two string are not same.  
strcmp(s1,s2)  
return a value:  
<0 when s1<s2  
=0 when s1=s2  
>0 when s1>s2  
The exact value returned in case of dissimilar strings is not defined. We only know  
that if s1<s2 then a negative value will be returned and if s1>s2 then a positive  
value will be returned.  
For example:

/\*String comparison…………………….\*/  
#include<stdio.h>  
#include<string.h>  
void main()  
{  
char str1[10],str2[10];  
printf(“Enter two strings:”);  
gets(str1);  
gets(str2);  
if(strcmp(str1,str2)==0)  
{  
printf(“String are same\n”);  
}  
else  
{  
printf(“String are not same\n”);  
} }  
**strcpy()**

This function is used to copying one string to another string. The function  
strcpy(str1,str2) copies str2 to str1 including the NULL character. Here str2 is the  
source string and str1 is the destination string.  
The old content of the destination string str1 are lost. The function returns a pointer  
to destination string str1.  
Example:-  
#include<stdio.h>  
#include<string.h>  
void main()  
{  
char str1[10],str2[10];  
printf(“Enter a string:”);  
scanf(“%s”,str2);  
strcpy(str1,str2);  
printf(“First string:%s\t\tSecond string:%s\n”,str1,str2);  
strcpy(str,”Delhi”);  
strcpy(str2,”Bangalore”);  
printf(“First string :%s\t\tSecond string:%s”,str1,str2);  
**strcat()**

This function is used to append a copy of a string at the end of the other string. If  
the first string is “”Purva” and second string is “Belmont” then after using this  
function the string becomes “PusvaBelmont”. The NULL character from str1 is  
moved and str2 is added at the end of str1. The 2nd string str2 remains unaffected.  
A pointer to the first string str1 is returned by the function.  
Example:-  
#include<stdio.h>  
#include<string.h>  
void main()  
{  
char str1[20],str[20];  
printf(“Enter two strings:”);  
gets(str1);  
gets(str2);  
strcat(str1,str2);  
printf(“First string:%s\t second string:%s\n”,str1,str2);  
strcat(str1,”-one”);  
printf(“Now first string is %s\n”,str1);  
}  
Output  
Enter two strings: data  
Base

FUNCTIONA function is a self contained block of codes or sub programs with a set of  
statements that perform some specific task or coherent task when it is called.  
It is something like to hiring a person to do some specific task like, every six  
months servicing a bike and hand over to it.  
Any ‘C’ program contain at least one function i.e main().  
There are basically two types of function those are  
1. Library function2. User defined functionThe user defined functions defined by the user according to its requirement  
System defined function can’t be modified, it can only read and can be used.  
These function are supplied with every C compiler  
Source of these library function are pre complied and only object code get used by  
the user by linking to the code by linker  
Here in system defined function description:  
Function definition : predefined, precompiled, stored in the library

**Function declaration** : In header file with or function prototype.  
**Function call** : By the programmer  
**User defined function**Syntax:-  
Return type name of function (type 1 arg 1, type2 arg2, type3 arg3)  
Return type function name argument list of the above syntax  
So when user gets his own function three thing he has to know, these are.  
**Function declaration  
Function definition  
Function call**These three things are represented like  
int function(int, int, int); /\*function declaration\*/  
main() /\* calling function\*/  
{  
function(arg1,arg2,arg3);  
}  
int function(type 1 arg 1,type2 arg2,type3, arg3) /\*function definition/\*  
{  
Local variable declaration;  
Statement;  
Return value;  
}

**Function declaration:**-  
Function declaration is also known as function prototype. It inform the compiler  
about three thing, those are name of the function, number and type of argument  
received by the function and the type of value returned by the function.  
While declaring the name of the argument is optional and the function prototype  
always terminated by the semicolon.  
**Function definition:-**Function definition consists of the whole description and code of the function.  
It tells about what function is doing what are its inputs and what are its out put  
It consists of two parts function header and function body  
Syntax:-  
return type function(type 1 arg1, type2 arg2, type3 arg3) /\*function header\*/  
{  
Local variable declaration;  
Statement 1;  
Statement 2;  
Return value  
}  
The return type denotes the type of the value that function will return and it is  
optional and if it is omitted, it is assumed to be int by default. The body of the  
function is the compound statements or block which consists of local variable  
declaration statement and optional return statement.

The local variable declared inside a function is local to that function only. It can’t  
be used anywhere in the program and its existence is only within this function.  
The arguments of the function **definition** are known as **formal arguments**.  
**Function Call**When the function get called by the calling function then that is called, function  
call. The compiler execute these functions when the semicolon is followed by the  
function name.  
Example:-  
function(arg1,arg2,arg3);  
The argument that are used inside the function call are called **actual argument**Ex:-  
int S=sum(a, b); //actual arguments  
**Actual argument**The arguments which are mentioned or used inside the function call is knows as  
actual argument and these are the original values and copy of these are actually  
sent to the called function  
It can be written as constant, expression or any function call like  
Function (x);  
Function (20, 30);  
Function (a\*b, c\*d);  
Function(2,3,sum(a, b));  
**Formal Arguments**The arguments which are mentioned in function definition are called formal  
arguments or dummy arguments.

These arguments are used to just hold the copied of the values that are sent by the  
calling function through the function call.  
These arguments are like other local variables which are created when the function  
call starts and destroyed when the function ends.  
The basic difference between the formal argument and the actual argument are  
**1)** The formal argument are declared inside the parenthesis where as the  
local variable declared at the beginning of the function block.  
**2)**. The **formal argument** are automatically initialized when the copy of actual  
arguments are passed while other local variable are assigned values through the  
statements.  
Order number and type of actual arguments in the function call should be match  
with the order number and type of the formal arguments.  
**Return type**It is used to return value to the calling function. It can be used in two way as  
return  
Or return(expression);  
Ex:- return (a);  
return (a\*b);  
return (a\*b+c);  
Here the 1st return statement used to terminate the function without returning any  
value  
Ex:- /\*summation of two values\*/  
int sum (int a1, int a2);  
main()

{  
int a,b;  
printf(“enter two no”);  
scanf(“%d%d”,&a,&b);  
int S=sum(a,b);  
printf(“summation is = %d”,s);  
}  
int sum(intx1,int y1)  
{  
int z=x1+y1;  
Return z;  
}  
**Advantage of function**By using function large and difficult program can be divided in to sub programs  
and solved. When we want to perform some task repeatedly or some code is to be  
used more than once at different place in the program, then function avoids this  
repeatition or rewritten over and over.  
Due to reducing size, modular function it is easy to modify and test  
**Notes**:-  
C program is a collection of one or more function.  
A function is get called when function is followed by the semicolon.  
A function is defined when a function name followed by a pair of curly braces

Any function can be called by another function even main() can be called by other  
function.  
main()  
{  
function1()  
}  
function1()  
{  
Statement;  
function2;  
}  
function 2()  
{ }  
So every function in a program must be called directly or indirectly by the main()  
function. A function can be called any number of times.  
A function can call itself again and again and this process is called **recursion**.  
A function can be called from other function **but** a function can’t be defined in  
another function

**Call by value and call by reference**There are two way through which we can pass the arguments to the function such  
as **call by** value and **call by reference**.  
**1. Call by value**In the call by value copy of the actual argument is passed to the formal argument  
and the operation is done on formal argument.  
When the function is called by ‘call by value’ method, it doesn’t affect content of  
the actual argument.  
Changes made to formal argument are local to block of called function so when the  
control back to calling function the changes made is vanish.  
Example:-  
main()  
{  
int x,y;  
change(int,int);

printf(“enter two values:\n”);  
scanf(“%d%d”,&x,&y);  
change(x ,y);  
printf(“value of x=%d and y=%d\n”,x ,y);  
}  
change(int a,int b);  
{  
int k;  
k=a;  
a=b;  
b=k;  
}  
Output: enter two values: 12  
23  
Value of x=12 and y=23  
**2. Call by reference**Instead of passing the value of variable, address or reference is passed and the  
function operate on address of the variable rather than value.  
Here formal argument is alter to the actual argument, it means formal arguments  
calls the actual arguments.  
Example:-  
void main()

{  
int a,b;  
change(int \*,int\*);  
printf(“enter two values:\n”);  
scanf(“%d%d”,&a,&b);  
change(&a,&b);  
printf(“after changing two value of a=%d and b=%d\n:”a,b);  
}  
change(int \*a, int \*b)  
{  
int k;  
k=\*a;  
\*a=\*b;  
\*b= k;  
printf(“value in this function a=%d and b=%d\n”,\*a,\*b);  
}  
Output: enter two values: 12  
32  
Value in this function a=32 and b=12  
After changing two value of a=32 and b=12  
So here instead of passing value of the variable, directly passing address of the  
variables. Formal argument directly access the value and swapping is possible even  
after calling a function.

RecursionWhen function calls itself (inside function body) again and again then it is  
called as recursive function. In recursion calling function and called function are  
same. It is powerful technique of writing complicated algorithm in easiest way.  
According to recursion problem is defined in term of itself. Here statement with in  
body of the function calls the same function and same times it is called as circular  
definition. In other words recursion is the process of defining something in form of  
itself.  
Syntax:  
main ()  
{  
rec(); /\*function call\*/  
rec();  
rec();  
Ex:- /\*calculate factorial of a no.using recursion\*/  
int fact(int);  
void main()

{  
int num;  
printf(“enter a number”);  
scanf(“%d”,&num);  
f=fact(num);  
printf(“factorial is =%d\n”f);  
}  
fact (int num)  
{  
If (num==0||num==1)  
return 1;  
else  
return(num\*fact(num-1));  
}

POINTERA pointer is a variable that store memory address or that contains address of  
another variable where addresses are the location number always contains whole  
number. So, pointer contain always the whole number. It is called pointer because  
it points to a particular location in memory by storing address of that location.  
SyntaxData type \*pointer name;Here \* before pointer indicate the compiler that variable declared as a pointer.  
e.g.  
int \*p1; //pointer to integer type  
float \*p2; //pointer to float type  
char \*p3; //pointer to character type  
When pointer declared, it contains garbage value i.e. it may point any value in the  
memory.

Two operators are used in the pointer i.e. **address operator(&)** and **indirection  
operator or dereference operator (\*).**Indirection operator gives the values stored at a particular address.  
Address operator cannot be used in any constant or any expression.  
Example:  
void main()  
{  
int i=105;  
int \*p;  
p=&i;  
t  
printf(“value of i=%d”,\*p);  
printf(“value of i=%d”,\*/&i);  
printf(“address of i=%d”,&i);  
printf(“address of i=%d”,p);  
printf(“address of p=%u”,&p);  
}  
**Pointer Expression  
Pointer assignment**int i=10;  
int \*p=&i;//value assigning to the pointer

Here declaration tells the compiler that P will be used to store the address of  
integer value or in other word P is a pointer to an integer and \*p reads the **value at  
the address contain in p.**P++;  
printf(“value of p=%d”);  
We can assign value of 1 pointer variable to other when their base type and data  
type is same or both the pointer points to the same variable as in the array.  
Int \*p1,\*p2;  
P1=&a[1];  
P2=&a[3];  
We can assign constant 0 to a pointer of any type for that symbolic constant  
‘**NULL**’ is used such as  
\*p=NULL;  
It means pointer doesn’t point to any valid memory location.  
**Pointer Arithmetic**Pointer arithmetic is different from ordinary arithmetic and it is perform relative to  
the data type(base type of a pointer).  
Example:-  
If integer pointer contain address of 2000 on incrementing we get address of 2002  
instead of 2001, because, size of the integer is of 2 bytes.  
Note:-  
When we move a pointer, somewhere else in memory by incrementing or  
decrement or adding or subtracting integer, it is not necessary that, pointer still  
pointer to a variable of same data, because, memory allocation to the variable are  
done by the compiler.

But in case of array it is possible, since there data are stored in a consecutive  
manner.  
Ex:-  
void main( )  
{  
static int a[ ]={20,30,105,82,97,72,66,102};  
int \*p,\*p1;  
P=&a[1];  
P1=&a[6];  
printf(“%d”,\*p1-\*p);  
printf(“%d”,p1-p);  
}  
**Arithmetic operation never perform on pointer are:  
addition, multiplication and division of two pointer.  
multiplication between the pointer by any number.  
division of pointer by any number  
-add of float or double value to the pointer.**Operation performed in pointer are:-  
/\* Addition of a number through pointer \*/  
Example  
int i=100;  
int \*p;

p=&i;  
p=p+2;  
p=p+3;  
p=p+9;  
ii /\* Subtraction of a number from a pointer’\*/  
Ex:-  
int i=22;  
\*p1=&a;  
p1=p1-10;  
p1=p1-2;  
iii- Subtraction of one pointer to another is possible when pointer variable point to  
an element of same type such as an array.  
Ex:-  
in tar[ ]={2,3,4,5,6,7};  
int \*ptr1,\*ptr1;  
ptr1=&a[3]; //2000+4  
ptr2=&a[6]; //2000+6

SructureIt is the collection of dissimilar data types or heterogenous data types grouped  
together. It means the data types may or may not be of same type.  
Structure declarationstruct tagname  
{  
Data type member1;  
Data type member2;  
Data type member3;  
………  
………  
Data type member n;  
};  
OR  
struct  
{  
Data type member1;  
Data type member2;

Data type member3;  
………  
………  
Data type member n;  
};  
OR  
struct tagname  
{  
struct element 1;  
struct element 2;  
struct element 3;  
………  
………  
struct element n;  
};  
Structure variable declaration;  
struct student  
{  
int age;  
char name[20];  
char branch[20];

}; struct student s;  
**Initialization of structure variable**Like primary variables structure variables can also be initialized when they are  
declared. Structure templates can be defined locally or globally. If it is local it can  
be used within that function. If it is global it can be used by all other functions of  
the program.  
We cant initialize structure members while defining the structure  
struct student  
{  
int age=20;  
char name[20]=”sona”;  
}s1;  
The above is **invalid.**A structure can be initialized as  
struct student  
{  
int age,roll;  
char name[20];  
} struct student s1={16,101,”sona”};  
struct student s2={17,102,”rupa”};  
If initialiser is less than no.of structure variable, automatically rest values are taken  
as zero.

**Accessing structure elements**Dot operator is used to access the structure elements. Its associativety is from left  
to right.  
structure variable ;  
s1.name[];  
s1.roll;  
s1.age;  
Elements of structure are stored in contiguous memory locations. Value of  
structure variable can be assigned to another structure variable of same type using  
assignment operator.  
Example:  
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
int roll, age;  
char branch;  
} s1,s2;  
printf(“\n enter roll, age, branch=”);  
scanf(“%d %d %c”, &s1.roll, &s1.age, &s1.branch);  
s2.roll=s1.roll;  
printf(“ students details=\n”);  
printf(“%d %d %c”, s1.roll, s1.age, s1.branch);  
printf(“%d”, s2.roll);

}  
**Unary, relational, arithmetic, bitwise operators** are not allowed within structure  
variables.

File handling

**File:** the file is a permanent storage medium in which we can store the data  
permanently.  
**Types of file can be handled**we can handle three type of file as  
**(1) sequential file  
(2) random access file  
(3) binary file  
File Operation  
opening a file:**Before performing any type of operation, a file must be opened and for this  
fopen() function is used.  
**syntax:**file-pointer=fopen(“FILE NAME ”,”Mode of open”);  
example:  
FILE \*fp=fopen(“ar.c”,”r”);  
If fopen() unable to open a file than it will return NULL to the file pointer.  
**File-pointer:** The file pointer is a pointer variable which can be store the address  
of a special file that means it is based upon the file pointer a file gets opened.  
**Declaration of a file pointer:-**FILE\* var;  
**Modes of open**The file can be open in three different ways as

**Read mode ’ r’/rt  
Write mode ’w’/wt  
Appened Mode ’a’/at  
Reading** a character from a file  
**getc()** is used to read a character into a file  
Syntax:  
character\_variable=getc(file\_ptr);  
**Writing** acharacter into a file  
**putc()** is used to write a character into a file  
**puts**(character-var,file-ptr);  
**ClOSING A FILE  
fclose()** function close a file.  
fclose(file-ptr);  
**fcloseall ()** is used to close all the opened file at a time  
**File Operation**The following file operation carried out the file  
(1)creation of a new file  
(3)writing a file  
(4)closing a file

Before performing any type of operation we must have to open the file.c, language  
communicate with file using A new type called **file pointer**.  
**Operation with fopen()**File pointer=fopen(“FILE NAME”,”mode of open”);  
If **fopen()** unable to open a file then it will return **NULL** to the file-pointer.

**Reading and writing a characters from/to a file  
fgetc()** is used for reading a character from the file  
**Syntax**:  
character variable= fgetc(file pointer);  
**fputc()** is used to writing a character to a file  
**Syntax**:  
fputc(character,file\_pointer);

/\*Program to copy a file to another\*/  
#include<stdio.h>  
void main()  
{  
FILE \*fs,\*fd;  
char ch;  
If(fs=fopen(“scr.txt”,”r”)==0)  
{  
printf(“sorry….The source file cannot be opened”);  
return;  
}  
If(fd=fopen(“dest.txt”,”w”)==0)  
{  
printf(“Sorry…..The destination file cannot be opened”);  
fclose(fs);  
return;  
}  
while(ch=fgets(fs)!=EOF)  
fputc(ch,fd);  
fcloseall();  
}

**Reading and writing a string from/to a file  
getw()** is used for reading a string from the file  
**Syntax**:  
gets(file pointer);  
**putw()** is used to writing a character to a file  
**Syntax**:  
fputs(integer,file\_pointer);  
#include<stdio.h>  
#include<stdlib.h>  
void main()  
{  
FILE \*fp;  
int word;  
/\*place the word in a file\*/  
fp=fopen(“dgt.txt”,”wb”);  
If(fp==NULL)  
{  
printf(“Error opening file”);  
exit(1);  
}  
word=94;  
putw(word,fp);  
If(ferror(fp))

printf(“Error writing to file\n”);  
else  
printf(“Successful write\n”);  
fclose(fp);  
/\*reopen the file\*/  
fp=fopen(“dgt.txt”,”rb”);  
If(fp==NULL)  
{  
printf(“Error opening file”);  
exit(1);  
}  
/\*extract the word\*/  
word=getw(fp);  
If(ferror(fp))  
printf(“Error reading file\n”);  
else  
printf(“Successful read:word=%d\n”,word);  
/\*clean up\*/  
fclose(fp);  
}