C – printf and scanf

- printf() and scanf() functions are inbuilt library functions in C which are available in C library by default. These functions are declared and related macros are defined in “stdio.h” which is a header file.
- We have to include “stdio.h” file as shown in below C program to make use of these printf() and scanf() library functions.

C – Data Types

- C data types are defined as the data storage format that a variable can store a data to perform a specific operation.
- Data types are used to define a variable before to use in a program.
- Size of variable, constant and array are determined by data types.

C – data types:

There are four data types in C language. They are,

S.no	Types	Data Types
1	Basic data types	int, char, float, double
2	Enumeration data type	enum
3	Derived data type	pointer, array, structure, union
4	Void data type	void

1. Basic data types in C:

1.1. Integer data type:

- Integer data type allows a variable to store numeric values.
- “int” keyword is used to refer integer data type.
- The storage size of int data type is 2 or 4 or 8 byte.
- It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for int data type.
- Like wise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for int datatype.
- int (2 byte) can store values from -32,768 to +32,767
- int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.
- If you want to use the integer value that crosses the above limit, you can go for “long int” and “long long int” for which the limits are very high.

Note:

- We can’t store decimal values using int data type.
- If we use int data type to store decimal values, decimal values will be truncated and we will get only whole number.
- In this case, float data type can be used to store decimal values in a variable.

1.2. Character data type:

- Character data type allows a variable to store only one character.
- Storage size of character data type is 1. We can store only one character using character data type.
- “char” keyword is used to refer character data type.
- For example, ‘A’ can be stored using char datatype. You can’t store more than one character using char data type.
- Please refer C – Strings topic to know how to store more than one characters in a variable.

1.3. Floating point data type:

Floating point data type consists of 2 types. They are,

1. float
2. double

1. float:

- Float data type allows a variable to store decimal values.
- Storage size of float data type is 4. This also varies depend upon the processor in the CPU as “int” data type.
- We can use up-to 6 digits after decimal using float data type.
- For example, 10.456789 can be stored in a variable using float data type.

2. double:

- Double data type is also same as float data type which allows up-to 10 digits after decimal.
- The range for double datatype is from 1E–37 to 1E+37.

1.3.1. sizeof() function in C:

sizeof() function is used to find the memory space allocated for each C data types.

C

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

int main()
{

   int a;
   char b;
   float c;
   double d;
   printf("Storage size for int data type:%d \n",sizeof(a));
   printf("Storage size for char data type:%d \n",sizeof(b));
   printf("Storage size for float data type:%d \n",sizeof(c));
   printf("Storage size for double data type:%d\n",sizeof(d));
   return 0;
}
```

Output:

```
Storage size for int data type:4
Storage size for char data type:1
Storage size for float data type:4
Storage size for double data type:8
```

1.3.2. Modifiers in C:

- The amount of memory space to be allocated for a variable is derived by modifiers.
- Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
- For example, storage space for int data type is 4 byte for 32 bit processor. We can increase the range by using long int which is 8 byte. We can decrease the range by using short int which is 2 byte.
- There are 5 modifiers available in C language. They are,

1. short
2. long
3. signed
4. unsigned
5. long long

- Below table gives the detail about the storage size of each C basic data type in 16 bit processor.
  Please keep in mind that storage size and range for int and float datatype will vary depend on the CPU processor (8,16, 32 and 64 bit)

S.No	C Data types	storage Size	Range
1	char	1	–127 to 127
2	int	2	–32,767 to 32,767
3	float	4	1E–37 to 1E+37 with six digits of precision
4	double	8	1E–37 to 1E+37 with ten digits of precision
5	long double	10	1E–37 to 1E+37 with ten digits of precision
6	long int	4	–2,147,483,647 to 2,147,483,647
7	short int	2	–32,767 to 32,767
8	unsigned short int	2	0 to 65,535
9	signed short int	2	–32,767 to 32,767
10	long long int	8	–(2power(63) –1) to 2(power)63 –1
11	signed long int	4	–2,147,483,647 to 2,147,483,647
12	unsigned long int	4	0 to 4,294,967,295

13	unsigned long long int	8	2(power)64 –1

2. Enumeration data type in C:

- Enumeration data type consists of named integer constants as a list.
- It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.
- Enum syntax in C:
  enum identifier [optional{ enumerator-list }];
- Enum example in C:
  enum month { Jan, Feb, Mar }; or
  /* Jan, Feb and Mar variables will be assigned to 0, 1 and 2 respectively by default */
  enum month { Jan = 1, Feb, Mar };
  /* Feb and Mar variables will be assigned to 2 and 3 respectively by default */
  enum month { Jan = 20, Feb, Mar };
  /* Jan is assigned to 20. Feb and Mar variables will be assigned to 21 and 22 respectively by default */

- The above enum functionality can also be implemented by "#define" preprocessor directive as given below. Above enum example is same as given below.
  #define Jan 20;
  #define Feb 21;
  #define Mar 22;

C – enum example program:

C

```
#include <stdio.h>

int main()
{
  enum MONTH { Jan = 0, Feb, Mar };

  enum MONTH month = Mar;

  if(month == 0)
    printf("Value of Jan");
  else if(month == 1)
    printf("Month is Feb");
  if(month == 2)
    printf("Month is Mar");
}
```

Output:

```
Month is March
```

3. Derived data type in C:

- Array, pointer, structure and union are called derived data type in C language.
- To know more about derived data types, please visit “C – Array“ , “C – Pointer” , “C – Structure” and “C – Union” topics in this tutorial.

4. Void data type in C:

- Void is an empty data type that has no value.
- This can be used in functions and pointers.
- Please visit “C – Function” topic to know how to use void data type in function with simple call by value and call by reference example programs.

C – Tokens and keywords

C tokens, Identifiers and Keywords are the basics in a C program. All are explained in this page with definition and simple example programs.

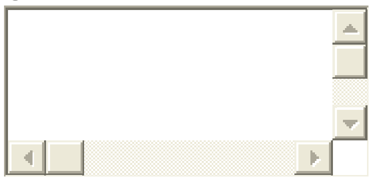
1. C tokens:

- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual units in a C program are known as C tokens.
- C tokens are of six types. They are,

1. Keywords (eg: int, while),
2. Identifiers (eg: main, total),
3. Constants (eg: 10, 20),
4. Strings (eg: “total”, “hello”),
5. Special symbols (eg: (), {}),
6. Operators (eg: +, /,-,*)

C tokens example program:

C



```
int main()
{
   int x, y, total;
   x = 10, y = 20;
   total = x + y;
   Printf (“Total = %d \n”, total);
}
```

where,

- main – identifier
- {,}, (,) – delimiter
- int – keyword
- x, y, total – identifier
- main, {, }, (,), int, x, y, total – tokens

Do you know how to use C token in real time application programs? We have given simple real time application programs where C token is used. You can refer the below C programs to know how to use C token in real time program.

Real time application C programs for your reference:

1. C program example – Real time Calculator program
2. C program example – Real time Bank Application program

2. Identifiers in C language:

- Each program elements in a C program are given a name called identifiers.
- Names given to identify Variables, functions and arrays are examples for identifiers. eg. x is a name given to integer variable in above program.

Rules for constructing identifier name in C:

1. First character should be an alphabet or underscore.
2. Succeeding characters might be digits or letter.
3. Punctuation and special characters aren't allowed except underscore.
4. Identifiers should not be keywords.

3. Keywords in C language:

- Keywords are pre-defined words in a C compiler.
- Each keyword is meant to perform a specific function in a C program.
- Since keywords are referred names for compiler, they can't be used as variable name.

C language supports 32 keywords which are given below. Click on each keywords below for detail description and example programs.

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	volatile

do	if	static	while

C – Constant

` `

- C Constants are also like normal variables. But, only difference is, their values can not be modified by the program once they are defined.
- Constants refer to fixed values. They are also called as literals
- Constants may be belonging to any of the data type.
- Syntax:
  const data_type variable_name; (or) const data_type *variable_name;

Types of C constant:

1. Integer constants
2. Real or Floating point constants
3. Octal & Hexadecimal constants
4. Character constants
5. String constants
6. Backslash character constants

S.no	Constant type	data type	Example
1	Integer constants	int unsigned int long int long long int	53, 762, -478 etc 5000u, 1000U etc 483,647 2,147,483,680
2	Real or Floating point constants	float doule	10.456789 600.123456789
3	Octal constant	int	013 /* starts with 0 */
4	Hexadecimal constant	int	0×90 /* starts with 0x */
5	character constants	char	‘A’ , ‘B’, ‘C’
6	string constants	char	“ABCD” , “Hai”

Rules for constructing C constant:

1. Integer Constants in C:

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can either be positive or negative.
- No commas or blanks are allowed within an integer constant.
- If no sign precedes an integer constant, it is assumed to be positive.

- The allowable range for integer constants is -32768 to 32767.

2. Real constants in C:

- A real constant must have at least one digit
- It must have a decimal point
- It could be either positive or negative
- If no sign precedes an integer constant, it is assumed to be positive.
- No commas or blanks are allowed within a real constant.

3. Character and string constants in C:

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.
- The maximum length of a character constant is 1 character.
- String constants are enclosed within double quotes.

4. Backslash Character Constants in C:

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol to make use of special function of them.
- Given below is the list of special characters and their purpose.

Backslash character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\”	Double quote
\’	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark
\N	Octal constant (N is an octal constant)
\XN	Hexadecimal constant (N – hex.dcml cnst)

How to use constants in a C program?

- We can define constants in a C program in the following ways.

1. By “const” keyword
2. By “#define” preprocessor directive

- Please note that when you try to change constant values after defining in C program, it will through error.

1. Example program using const keyword in C:

C

```
#include <stdio.h>

void main()
{
   const int  height = 100;                /*int constant*/
   const float number = 3.14;              /*Real constant*/
   const char letter = 'A';                /*char constant*/
   const char letter_sequence[10] = "ABC"; /*string constant*/
   const char backslash_char = '\?';       /*special char cnst*/

   printf("value of height    : %d \n", height );
   printf("value of number : %f \n", number );
   printf("value of letter : %c \n", letter );
   printf("value of letter_sequence : %s \n", letter_sequence);
   printf("value of backslash_char  : %c \n", backslash_char);
}
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

2. Example program using #define preprocessor directive in C:

C

```
#include <stdio.h>

#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\?'

void main()
{

   printf("value of height    : %d \n", height );
   printf("value of number : %f \n", number );
   printf("value of letter : %c \n", letter );
   printf("value of letter_sequence : %s \n", letter_sequence);
   printf("value of backslash_char  : %c \n", backslash_char);
```

17
18 }

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

C – Variable

- C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable.
- The value of the C variable may get change in the program.
- C variable might be belonging to any of the data type like int, float, char etc.

Rules for naming C variable:

1. Variable name must begin with letter or underscore.
2. Variables are case sensitive
3. They can be constructed with digits, letters.
4. No special symbols are allowed other than underscore.
5. sum, height, _value are some examples for variable name

Declaring & initializing C variable:

- Variables should be declared in the C program before to use.
- Memory space is not allocated for a variable while declaration. It happens only on variable definition.
- Variable initialization means assigning a value to the variable.

S.No	Type	Syntax	Example
1	Variable declaration	data_type variable_name;	int x, y, z; char flat, ch;
2	Variable initialization	data_type variable_name = value;	int x = 50, y = 30; char flag = ‘x’, ch=’l’;

There are three types of variables in C program They are,

1. Local variable

2. Global variable
3. Environment variable

1. Example program for local variable in C:

- The scope of local variables will be within the function only.
- These variables are declared within the function and can’t be accessed outside the function.
- In the below example, m and n variables are having scope within the main function only. These are not visible to test function.
- Like wise, a and b variables are having scope within the test function only. These are not visible to main function.

C

```c
#include<stdio.h>

void test();

int main()
{
     int m = 22, n = 44;
     // m, n are local variables of main function

     /*m and n variables are having scope
          within this main function only.
          These are not visible to test funtion.*/
     /* If you try to access a and b in this function,
        you will get 'a' undeclared and 'b' undeclared
        error */

     printf("\nvalues : m = %d and n = %d", m, n);

     test();

}

void test()
{
     int a = 50, b = 80;
     // a, b are local variables of test function

     /*a and b variables are having scope
          within this test function only.
          These are not visible to main function.*/
     /* If you try to access m and n in this function,
        you will get 'm' undeclared and 'n' undeclared
        error */

     printf("\nvalues : a = %d and b = %d", a, b);
}
```

Output:

```
values : m = 22 and n = 44
values : a = 50 and b = 80
```

2. Example program for global variable in C:

- The scope of global variables will be throughout the program. These variables can be accessed from anywhere in the program.
- This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.

C

```c
#include<stdio.h>

void test();

int m = 22, n = 44;
int a = 50, b = 80;

int main()
{

  printf("All variables are accessed from main function");

  printf("\nvalues : m= %d : n= %d : a= %d : b= %d",m,n,a,b);

  test();

}

void test()
{

  printf("\n\nAll variables are accessed from" \
         " test function");

  printf("\nvalues : m= %d : n= %d : a= %d : b= %d",m,n,a,b);
}
```

Output:

```
All variables are accessed from main function
values : m = 22 : n = 44 : a = 50 : b = 80

All variables are accessed from test function
values : m = 22 : n = 44 : a = 50 : b = 80
```

3. Environment variables in C:

- Environment variable is a variable that will be available for all C applications and C programs.
- We can access these variables from anywhere in a C program without declaring and initializing in an application or C program.
- The inbuilt functions which are used to access, modify and set these environment variables are called environment functions.
- There are 3 functions which are used to access, modify and assign an environment variable in C. They are,

  1. setenv()
  2. getenv()
  3. putenv()

Example program for getenv() function in C:

This function gets the current value of the environment variable. Let us assume that environment variable DIR is assigned to “/usr/bin/test/”.

C

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

int main()
{
  printf("Directory = %s\n", getenv("DIR"));
  return 0;
}
```

Output:

```
/usr/bin/test/
```

Example program for setenv() function in C:

This function sets the value for environment variable. Let us assume that environment variable “FILE” is to be assigned “/usr/bin/example.c”

C

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

```
int main()
{
   setenv("FILE","/usr/bin/example.c",50);
   printf("File = %s\n", getenv("FILE"));
   return 0;
}
```

Output:

```
File = /usr/bin/example.c
```

Example program for putenv() function in C:

This function modifies the value for environment variable. Below example program shows that how to modify an existing environment variable value.

C

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   setenv("DIR","/usr/bin/example/",50);
   printf("Directory name before modifying = " \
          "%s\n", getenv("DIR"));

   putenv("DIR=/usr/home/");
   printf("Directory name after modifying = " \
            "%s\n", getenv("DIR"));
   return 0;
}
```

Output:

```
Directory name before modifying = /usr/bin/example/
Directory name after modifying = /usr/home/
```

Difference between variable declaration & definition in C:

S.no	Variable declaration	Variable definition
1	Declaration tells the compiler about data type and size of the variable.	Definition allocates memory for the variable.
2	Variable can be declared many times in a program.	It can happen only one time for a variable in a program.

3	The assignment of properties and identification to a variable.	Assignments of storage space to a variable.

C – Operators and Expressions

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- Consider the expression A + B * 5. where, +, * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression.

Types of C operators:

C language offers many types of operators. They are,

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

Continue on types of C operators:

- Click on each operators name below for detail description and example programs.

S.no	Types of Operators	Description
1	**Arithmetic operators**	These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus
2	**Assignment operators**	These are used to assign the values for the variables in C programs.
3	**Relational operators**	These operators are used to compare the value of two variables.
4	**Logical operators**	These operators are used to perform logical operations on the given two variables.
5	**Bit wise operators**	These operators are used to perform bit operations on given two variables.
6	**Conditional (ternary) operators**	Conditional operators return one value if condition is true and returns another value is condition is false.

7	**Increment/decrement operators**	These operators are used to either increase or decrease the value of the variable by one.
8	**Special operators**	&, *, sizeof( ) and ternary operators.

C – Arithmetic Operators

Arithmetic Operators in C:

- C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs.

S.no	Arithmetic Operators	Operation	Example
1	+	Addition	A+B
2	-	Subtraction	A-B
3	*	multiplication	A*B
4	/	Division	A/B
5	%	Modulus	A%B

Example program for C arithmetic operators:

- In this example program, two values “40″ and “20″ are used to perform arithmetic operations such as addition, subtraction, multiplication, division, modulus and output is displayed for each operation.

C

```
#include <stdio.h>
int main()
{
   int a=40,b=20, add,sub,mul,div,mod;

   add = a+b;
   sub = a-b;
   mul = a*b;
   div = a/b;
   mod = a%b;

   printf("Addition of a, b is   : %d\n", add);
   printf("Subtraction of a, b is   : %d\n", sub);
   printf("Multiplication of a, b is   : %d\n", mul);
   printf("Division of a, b is   : %d\n", div);
   printf("Modulus of a, b is   : %d\n", mod);
}
```

Output:

```
Addition of a, b is : 60
```

```
Subtraction of a, b is : 20
Multiplication of a, b is : 800
Division of a, b is : 2
Modulus of a, b is : 0
```

S.no	Types of Operators	Description
1	**Arithmetic operators**	These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus
2	Assignment operators	These are used to assign the values for the variables in C programs.
3	Relational operators	These operators are used to compare the value of two variables.
4	Logical operators	These operators are used to perform logical operations on the given two variables.
5	Bit wise operators	These operators are used to perform bit operations on given two variables.
6	Conditional (ternary) operators	Conditional operators return one value if condition is true and returns another value is condition is false.
7	Increment / decrement operators	These operators are used to either increase or decrease the value of the variable by one.
8	Special operators	&, *, sizeof( ) and ternary operators.

C – Assignment Operators

Assignment operators in C:

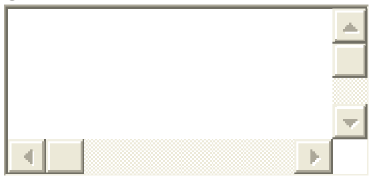
- In C programs, values for the variables are assigned using assignment operators.
- For example, if the value “10″ is to be assigned for the variable “sum”, it can be assigned as “sum = 10;”
- Other assignment operators in C language are given below.

Operators		Example	Explanation
Simple assignment operator	=	sum=10	10 is assigned to variable sum
Compound assignment operators	+=	sum+=10	This is same as sum=sum+10
	-=	sum-=10	This is same as sum = sum-10
	*=	sum*=10	This is same as sum = sum*10
	/+	sum/=10	This is same as sum = sum/10
	%=	sum%=10	This is same as sum = sum%10
	&=	sum&=10	This is same as sum = sum&10
	^=	sum^=10	This is same as sum = sum^10

Example program for C assignment operators:

- In this program, values from 0 – 9 are summed up and total “45″ is displayed as output.
- Assignment operators such as “=” and “+=” are used in this program to assign the values and to sum up the values.

C



```
# include <stdio.h>
int main()
{
   int Total=0,i;
   for(i=0;i<10;i++)
   {
      Total+=i; // This is same as Total = Toatal+i
   }
   printf("Total = %d", Total);
}
```

Output:

```
Total = 45
```

C – Decision Control statement

- In decision control statements (C if else and nested if), group of statements are executed when condition is true. If condition is false, then else part statements are executed.
- There are 3 types of decision making control statements in C language. They are,

1. if statements
2. if else statements
3. nested if statements

“If”, “else” and “nested if” decision control statements in C:

- Syntax for each C decision control statements are given in below table with description.

Decision control statements	Syntax	Description

if	if (condition)
	{ Statements; }
		In these type of statements, if condition is true, then respective block of code is executed.

if…else	if (condition)
	{ Statement1; Statement2;}
	else
		In these type of statements, group of statements are executed when condition is true. If condition is false, then else part

	{ Statement3; Statement4; }	statements are executed.
nested if	if (condition1){ Statement1; } else if (condition2) { Statement2; } else Statement 3;	If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed.

Example program for if statement in C:

In “if” control statement, respective block of code is executed when condition is true.

C

```
int main()
{
  int m=40,n=40;
  if (m == n)
  {
      printf("m and n are equal");
  }
}
```

Output:

```
m and n are equal
```

Example program for if else statement in C:

In C if else control statement, group of statements are executed when condition is true. If condition is false, then else part statements are executed.

C

```
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m == n) {
      printf("m and n are equal");
  }
  else {
       printf("m and n are not equal");
  }
```

```c

}
```

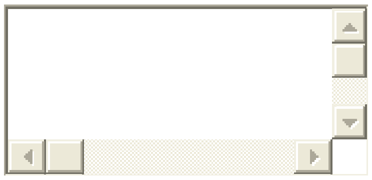
Output:

```
m and n are not equal
```

Example program for nested if statement in C:

- In "nested if" control statement, if condition 1 is false, then condition 2 is checked and statements are executed if it is true.
- If condition 2 also gets failure, then else part is executed.

C



```c
#include <stdio.h>
int main()
{
 int m=40,n=20;
 if (m>n) {
     printf("m is greater than n");
 }
 else if(m<n) {
       printf("m is less than n");
 }
 else {
       printf("m is equal to n");
 }
}
```

Output:

```
m is greater than n
```

C – Loop control statements

Loop control statements in C are used to perform looping operations until the given condition is true. Control comes out of the loop statements once condition becomes false.

Types of loop control statements in C:

There are 3 types of loop control statements in C language. They are,

1. for
2. while
3. do-while
4. Syntax for each C loop control statements are given in below table with description.

S.no	Loop Name	Syntax	Description

1	for	for (exp1; exp2; expr3)
		{ statements; }
			Where,
			exp1 – variable initialization
			(Example: i=0, j=2, k=3)
			exp2 – condition checking
			(Example: i>5, j<3, k=3)
			exp3 – increment/decrement
			(Example: ++i, j–, ++k)

2	while	while (condition)
		{ statements; }
			where,
			condition might be a>5, i<10

3	do while	do { statements; }
		while (condition);
			where,
			condition might be a>5, i<10

Example program (for loop) in C:

In for loop control statement, loop is executed until condition becomes false.

C

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
      printf("%d ",i);
  }

}
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Example program (while loop) in C:

In while loop control statement, loop is executed until condition becomes false.

C

```
#include <stdio.h>

int main()
{
  int i=3;

  while(i<10)
  {
      printf("%d\n",i);
      i++;
  }

}
```

Output:

```
3 4 5 6 7 8 9
```

Example program (do while loop) in C:

In do..while loop control statement, while loop is executed irrespective of the condition for first time. Then 2nd time onwards, loop is executed until condition becomes false.

C

```
#include <stdio.h>

int main()
{
  int i=1;

  do
  {
      printf("Value of i is %d\n",i);
      i++;
  }while(i<=4 && i>=2);

}
```

Output:

```
Value of i is 1
Value of i is 2
Value of i is 3
Value of i is 4
```

Difference between while & do while loops in C:

S.no	while	do while
1	Loop is executed only when condition is true.	Loop is executed for first time irrespective of the condition. After executing while loop for first time, then condition is checked.

C – Case control statements

The statements which are used to execute only specific block of statements in a series of blocks are called case control statements.

There are 4 types of case control statements in C language. They are,

1. switch
2. break
3. continue
4. goto

1. switch case statement in C:

- Switch case statements are used to execute only specific case statements based on the switch expression.
- Below is the syntax for switch case statement.

```
switch (expression)
{
        case label1:   statements;
                        break;
        case label2:   statements;
                        break;
        default:        statements;
                        break;
}
```

Example program for switch..case statement in C:

C

```
#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
```

2. break statement in C:

- Break statement is used to terminate the while loops, switch case loops and for loops from the subsequent execution.
- Syntax: break;

Example program for break statement in C:

C

```
#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
```

3. Continue statement in C:

- Continue statement is used to continue the ` iteration of for loop, while loop and do-while loops. So, the remaining statements are skipped within the loop for that particular iteration.
- Syntax : continue;

Example program for continue statement in C:

C

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
     if(i==5 || i==6)
     {
        printf("\nSkipping %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
```

4. goto statement in C:

- goto statements is used to transfer the normal flow of a program to the specified label in the program.
- Below is the syntax for goto statement in C.

```
{
       …….
       go to label;
       …….
       …….
       LABEL:
       statements;
}
```

Example program for goto statement in C:

C

```c
#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”
```

C – Type Qualifiers

- C – type qualifiers : The keywords which are used to modify the properties of a variable are called type qualifiers.

Types of C type qualifiers:

There are two types of qualifiers available in C language. They are,

1. const
2. volatile

1. const keyword:

- Constants are also like normal variables. But, only difference is, their values can't be modified by the program once they are defined.
- They refer to fixed values. They are also called as literals.
- They may be belonging to any of the data type.
- Syntax:
  const data_type variable_name; (or) const data_type *variable_name;
- Please refer **C – Constants** topic in this tutorial for more details on const keyword.

2. volatile keyword:

- When a variable is defined as volatile, the program may not change the value of the variable explicitly.
- But, these variable values might keep on changing without any explicit assignment by the program. These types of qualifiers are called volatile.
- For example, if global variable's address is passed to clock routine of the operating system to store the system time, the value in this address keep on changing without any assignment by the program. These variables are named as volatile variable.
- Syntax:
  volatile data_type variable_name; (or) volatile data_type *variable_name;

C – Storage Class Specifiers

Storage class specifiers in C language tells the compiler where to store a variable, how to store the variable, what is the initial value of the variable and life time of the variable.

Syntax: storage_specifier data_type variable _name

Types of Storage Class Specifiers in C:

There are 4 storage class specifiers available in C language. They are,

1. auto
2. extern
3. static
4. register

S.No.	Storage Specifier	Storage place	Initial / default	Scope	Life

			value		
1	**auto**	CPU Memory	Garbage value	local	Within the function only.
2	**extern**	CPU memory	Zero	Global	Till the end of the main program. Variable definition might be anywhere in the C program
3	**static**	CPU memory	Zero	local	Retains the value of the variable between different function calls.
4	**register**	Register memory	Garbage value	local	Within the function

Note:

- For faster access of a variable, it is better to go for register specifiers rather than auto specifiers.
- Because, register variables are stored in register memory whereas auto variables are stored in main CPU memory.
- Only few variables can be stored in register memory. So, we can use variables as register that are used very often in a C program.

Example program for auto variable in C:

The scope of this auto variable is within the function only. It is equivalent to local variable. All local variables are auto variables by default.

C

```
#include<stdio.h>
void increment(void);

int main()
{
  increment();
  increment();
  increment();
  increment();
  return 0;
}

void increment(void)
{
  auto int i = 0 ;
  printf ( "%d ", i ) ;
  i++;
}
```

Output:

```
0 0 0 0
```

Example program for static variable in C:

Static variables retain the value of the variable between different function calls.

C

```
//C static example
#include<stdio.h>
void increment(void);

int main()
{
  increment();
  increment();
  increment();
  increment();
  return 0;
}

void increment(void)
{
  static int i = 0 ;
  printf ( "%d ", i ) ;
  i++;
}
```

Output:

```
0 1 2 3
```

Example program for extern variable in C:

The scope of this extern variable is throughout the main program. It is equivalent to global variable. Definition for extern variable might be anywhere in the C program.

C

```
#include<stdio.h>
int x = 10 ;
int main( )
{
```

```c
  extern int y ;
  printf ( "The value of x is %d \n", x ) ;
  printf ( "The value of y is %d",y ) ;
  return 0;
}
int y = 50 ;
```

Output:

```
The value of x is 10
The value of y is 50
```

Example program for register variable in C:

- Register variables are also local variables, but stored in register memory. Whereas, auto variables are stored in main CPU memory.
- Register variables will be accessed very faster than the normal variables since they are stored in register memory rather than main memory.
- But, only limited variables can be used as register since register size is very low. (16 bits, 32 bits or 64 bits)

C

```c
#include <stdio.h>

int main()
{
  register int i;
  int arr[5];          // declaring array
  arr[0] = 10;         // Initializing array
  arr[1] = 20;
  arr[2] = 30;
  arr[3] = 40;
  arr[4] = 50;
  for (i=0;i<5;i++)
  {
    // Accessing each variable
    printf("value of arr[%d] is %d \n", i, arr[i]);
  }
  return 0;
}
```

Output:

```
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
```

C – Array

C Array is a collection of variables belongings to the same data type. You can store group of data of same data type in an array.

- Array might be belonging to any of the data types
- Array size must be a constant value.
- Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
- It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

Example for C Arrays:

- int a[10]; // integer array
- char b[10]; // character array i.e. string

Types of C arrays:

There are 2 types of C arrays. They are,

1. One dimensional array
2. Multi dimensional array

- Two dimensional array
- Three dimensional array, four dimensional array etc…

1. One dimensional array in C:

- Syntax : data-type arr_name[array_size];

Array declaration	Array initialization	Accessing array
Syntax: data_type arr_name [arr_size];	data_type arr_name [arr_size]= (value1, value2, value3,….);	arr_name[index];
int age [5];	int age[5]={0, 1, 2, 3, 4, 5};	age[0]; /*0 is accessed*/ age[1]; /*1 is accessed*/ age[2]; /*2 is accessed*/

char str[10];	char str[10]={'H','a','i'}; (or)
	char str[0] = 'H';
	char str[1] = 'a';
	char str[2] = 'i;
		str[0]; /*H is accessed*/
		str[1]; /*a is accessed*/
		str[2]; /* i is accessed*/

Example program for one dimensional array in C:

C

```
#include<stdio.h>

int main()
{
   int i;
   int arr[5] = {10,20,30,40,50};
   // declaring and Initializing array in C
   //To initialize all array elements to 0, use int arr[5]={0};
   /* Above array can be initialized as below also
      arr[0] = 10;
      arr[1] = 20;
      arr[2] = 30;
      arr[3] = 40;
      arr[4] = 50;
   */
   for (i=0;i<5;i++)
   {
      // Accessing each variable
      printf("value of arr[%d] is %d \n", i, arr[i]);
   }
}
```

Output:

```
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
```

2. Two dimensional array in C:

- Two dimensional array is nothing but array of array.
- syntax : data_type array_name[num_of_rows][num_of_column]

S.no	Array declaration	Array initialization	Accessing array
1	Syntax: data_type arr_name [num_of_rows][num_of_column];	data_type arr_name[2][2] = {{0,0},{0,1},{1,0},{1,1}};	arr_name[index];

2	Example:
	int arr[2][2];
		int arr[2][2] = {1,2, 3, 4};	arr [0] [0] = 1;
			arr [0]]1] = 2;
			arr [1][0] = 3;
			arr [1] [1] = 4;

Example program for two dimensional array in C:

C

```
#include<stdio.h>
int main()
{
   int i,j;
   // declaring and Initializing array
   int arr[2][2] = {10,20,30,40};
   /* Above array can be initialized as below also
      arr[0][0] = 10;   // Initializing array
      arr[0][1] = 20;
      arr[1][0] = 30;
      arr[1][1] = 40;
   */
   for (i=0;i<2;i++)
   {
      for (j=0;j<2;j++)
      {
         // Accessing variables
         printf("value of arr[%d] [%d] : %d\n",i,j,arr[i][j]);
      }
   }
}
```

Output:

```
value of arr[0] [0] is 10
value of arr[0] [1] is 20
value of arr[1] [0] is 30
value of arr[1] [1] is 40
```

C – String

- C Strings are nothing but array of characters ended with null character (‘\0’).
- This null character indicates the end of the string.
- Strings are always enclosed by double quotes. Whereas, character is enclosed by single quotes in C.

C – Pointer

- C Pointer is a variable that stores/points the address of the another variable.
- C Pointer is used to allocate memory dynamically i.e. at run time.
- The variable might be any of the data type such as int, float, char, double, short etc.

  Syntax : data_type *var_name;

  Example : int *p; char *p;

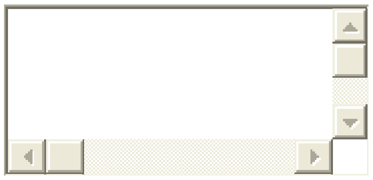
- Where, * is used to denote that “p” is pointer variable and not a normal variable.

Key points to remember about pointers in C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- & symbol is used to get the address of the variable.
- * symbol is used to get the value of the variable that the pointer is pointing to.
- If pointer is assigned to NULL, it means it is pointing to nothing.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- The size of any pointer is 2 byte (for 16 bit compiler).

Example program for pointer in C:

C



```
#include <stdio.h>

int main()
{
       int *ptr, q;
       q = 50;
       /* address of q is assigned to ptr       */
       ptr = &q;
       /* display q's value using ptr variable */
       printf("%d", *ptr);
       return 0;
}
```

Output:

```
50
```

C – Function

C functions are basic building blocks in a program. All C programs are written using functions to improve re-usability, understandability and to keep track on them. You can learn below concepts of C functions in this section in detail.

1. What is C function?
2. Uses of C functions
3. C function declaration, function call and definition with example program
4. How to call C functions in a program?

- Call by value

- Call by reference

5. C function arguments and return values

- C function with arguments and with return value
- C function with arguments and without return value
- C function without arguments and without return value
- C function without arguments and with return value

6. Types of C functions

- Library functions in C
- User defined functions in C
- Creating/Adding user defined function in C library

7. Command line arguments in C
8. Variable length arguments in C

1. What is C function?

A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

2. Uses of C functions:

- C functions are used to avoid rewriting same logic/code again and again in a program.
- There is no limit in calling C functions to make use of same functionality wherever required.
- We can call functions any number of times in a program and from any place in a program.
- A large C program can easily be tracked when it is divided into functions.
- The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

3. C function declaration, function call and function definition:

There are 3 aspects in each C function. They are,

- Function declaration or prototype - This informs compiler about the function name, function parameters and return value's data type.
- Function call – This calls the actual function
- Function definition – This contains all the statements to be executed.

S.no	C function aspects	syntax
1	function definition	return_type function_name (arguments list) { Body of function; }
2	function call	function_name (arguments list);
3	function declaration	return_type function_name (argument list);

Simple example program for C function:

- As you know, functions should be declared and defined before calling in a C program.
- In the below program, function "square" is called from main function.

- The value of “m” is passed as argument to the function “square”. This value is multiplied by itself in this function and multiplied value “p” is returned to main function from function “square”.

C

```c
#include<stdio.h>
// function prototype, also called function declaration
float square ( float x );

// main function, program starts from here
int main( )
{

        float m, n ;
        printf ( "\nEnter some number for finding square \n");
        scanf ( "%f", &m ) ;
        // function call
        n = square ( m ) ;
        printf ( "\nSquare of the given number %f is %f",m,n );

}

float square ( float x )   // function definition
{
        float p ;
        p = x * x ;
        return ( p ) ;
}
```

Output:

```
Enter some number for finding square
2
Square of the given number 2.000000 is 4.000000
```

4. How to call C functions in a program?

There are two ways that a C function can be called from a program. They are,

1. Call by value
2. Call by reference

1. Call by value:

- In call by value method, the value of the variable is passed to the function as parameter.
- The value of the actual parameter can not be modified by formal parameter.
- Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

Note:

- Actual parameter – This is the argument which is used in function call.
- Formal parameter – This is the argument which is used in function definition

Example program for C function (using call by value):

- In this program, the values of the variables “m” and “n” are passed to the function “swap”.
- These values are copied to formal parameters “a” and “b” in swap function and used.

C

```c
#include<stdio.h>
// function prototype, also called function declaration
void swap(int a, int b);

int main()
{
   int m = 22, n = 44;
   // calling swap function by value
   printf(" values before swap  m = %d \nand n = %d", m, n);
   swap(m, n);
}

void swap(int a, int b)
{
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
   printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

Output:

```
values before swap m = 22
and n = 44
values after swap m = 44
and n = 22
```

2. Call by reference:

- In call by reference method, the address of the variable is passed to the function as parameter.
- The value of the actual parameter can be modified by formal parameter.
- Same memory is used for both actual and formal parameters since only address is used by both parameters.

Example program for C function (using call by reference):

- In this program, the address of the variables “m” and “n” are passed to the function “swap”.
- These values are not copied to formal parameters “a” and “b” in swap function.
- Because, they are just holding the address of those variables.

- This address is used to access and change the values of the variables.

C

```c
#include<stdio.h>
// function prototype, also called function declaration
void swap(int *a, int *b);

int main()
{

   int m = 22, n = 44;
   //  calling swap function by reference
   printf("values before swap m = %d \n and n = %d",m,n);
   swap(&m, &n);

}

void swap(int *a, int *b)
{
   int tmp;
   tmp = *a;
   *a = *b;
   *b = tmp;
   printf("\n values after swap a = %d \nand b = %d", *a, *b);
}
```

Output:

```
values before swap m = 22
and n = 44
values after swap a = 44
and b = 22
```

C – Argument & return value

All C functions can be called either with arguments or without arguments in a C program. These functions may or may not return values to the calling function. Now, we will see simple example C programs for each one of the below.

1. C function with arguments (parameters) and with return value
2. C function with arguments (parameters) and without return value
3. C function without arguments (parameters) and without return value
4. C function without arguments (parameters) and with return value

S.no	**C function**	**syntax**

1	with arguments and with return values	int function (int); // function declaration
		function (a); // function call
		int function(int a) // function definition
		{statements; return a;}

2	with arguments and without return values	void function (int); // function declaration
		function(a); // function call
		void function(int a) // function definition
		{statements;}

3	without arguments and without return values	void function(); // function declaration
		function(); // function call
		void function() // function definition
		{statements;}

4	without arguments and with return values	int function ( ); // function declaration
		function ( ); // function call
		int function( ) // function definition
		{statements; return a;}

Note:

- If the return data type of a function is “void”, then, it can’t return any values to the calling function.
- If the return data type of the function is other than void such as “int, float, double etc”, then, it can return values to the calling function.

1. Example program for with arguments & with return value:

In this program, integer, array and string are passed as arguments to the function. The return type of this function is “int” and value of the variable “a” is returned from the function. The values for array and string are modified inside the function itself.

C

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

int function(int, int[], char[]);

int main()
{
     int i, a = 20;
     int arr[5] = {10,20,30,40,50};
     char str[30] = "\"fresh2refresh\"";

     printf("    ***values before modification***\n");
     printf("value of a is %d\n",a);

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

```
    {
      // Accessing each variable
      printf("value of arr[%d] is %d\n",i,arr[i]);
    }
    printf("value of str is %s\n",str);

    printf("\n    ***values after modification***\n");

    a= function(a, &arr[0], &str[0]);

    printf("value of a is %d\n",a);

    for (i=0;i<5;i++)
    {
      // Accessing each variable
      printf("value of arr[%d] is %d\n",i,arr[i]);
    }
    printf("value of str is %s\n",str);

    return 0;
}

int function(int a, int *arr, char *str)
{
   int i;

   a = a+20;
   arr[0] = arr[0]+50;
   arr[1] = arr[1]+50;
   arr[2] = arr[2]+50;
   arr[3] = arr[3]+50;
   arr[4] = arr[4]+50;
   strcpy(str,"\"modified string\"");

   return a;

}
```

Output:

```
***values before modification***
value of a is 20
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
value of str is “fresh2refresh”
***values after modification***
value of a is 40
value of arr[0] is 60
value of arr[1] is 70
value of arr[2] is 80
value of arr[3] is 90
value of arr[4] is 100
value of str is “modified string”
```

2. Example program for with arguments & without return value:

In this program, integer, array and string are passed as arguments to the function. The return type of this function is “void” and no values can be returned from the function. All the values of integer, array and string are manipulated and displayed inside the function itself.

C

```c
#include<stdio.h>

void function(int, int[], char[]);

int main()
{
    int a = 20;
    int arr[5] = {10,20,30,40,50};
    char str[30] = "\"fresh2refresh\"";

    function(a, &arr[0], &str[0]);
    return 0;
}

void function(int a, int *arr, char *str)
{
  int i;

  printf("value of a is %d\n\n",a);

    for (i=0;i<5;i++)
    {
       // Accessing each variable
       printf("value of arr[%d] is %d\n",i,arr[i]);
    }
  printf("\nvalue of str is %s\n",str);
}
```

Output:

```
value of a is 20

value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50

value of str is “fresh2refresh”
```

3. Example program for without arguments & without return value:

In this program, no values are passed to the function “test” and no values are returned from this function to main function.

C

```c
#include<stdio.h>

void test();

int main()
{
   test();
   return 0;
}

void test()
{
     int a = 50, b = 80;
     printf("\nvalues : a = %d and b = %d", a, b);
}
```

Output:

```
values : a = 50 and b = 80
```

4. Example program for without arguments & with return value:

In this program, no arguments are passed to the function “sum”. But, values are returned from this function to main function. Values of the variable a and b are summed up in the function “sum” and the sum of these value is returned to the main function.

C

```c
#include<stdio.h>

int sum();

int main()
{
   int addition;
   addition = sum();
   printf("\nSum of two given values = %d", addition);
   return 0;
}

int sum()
{
      int a = 50, b = 80, sum;

      sum = a + b;
      return sum;
}
```

Output:

```
Sum of two given values = 130
```

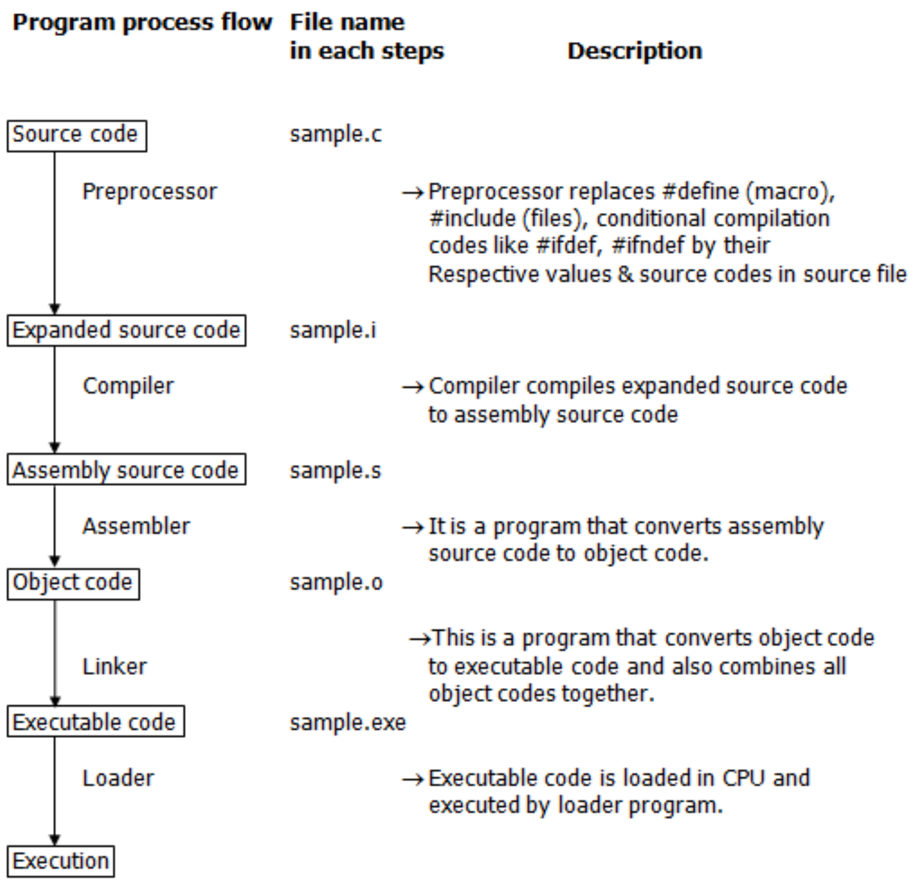
C – Preprocessor directives

C Preprocessor directives:

- Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing.
- Commands used in preprocessor are called preprocessor directives and they begin with "#" symbol.
- Below is the list of preprocessor directives that C language offers.

S.no	Preprocessor	Syntax	Description
1	Macro	#define	This macro defines constant value and can be any of the basic data types.
2	Header file inclusion	#include <file_name>	The source code of the file "file_name" is included in the main program at the specified place
3	Conditional compilation	#ifdef, #endif, #if, #else, #ifndef	Set of commands are included or excluded in source program before compilation with respect to the condition
4	Other directives	#undef, #pragma	#undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after main function in a C program

A program in C language involves into different processes. Below diagram will help you to understand all the processes that a C program comes across.


Program process flow
File name in each steps
Description
Source code
sample.c
Preprocessor
→Preprocessor replaces #define (macro), #include (files), conditional compilation codes like #ifdef, #ifndef by their Respective values & source codes in source file
Expanded source code
sample.i
Compiler
→Compiler compiles expanded source code to assembly source code
Assembly source code
sample.s
Assembler
→It is a program that converts assembly source code to object code.
Object code
sample.o
Linker
→This is a program that converts object code to executable code and also combines all object codes together.
Executable code
sample.exe
Loader
→Executable code is loaded in CPU and executed by loader program.
Execution

Example program for #define, #include preprocessors in C:

- #define - This macro defines constant value and can be any of the basic data types.
- #include <file_name> - The source code of the file "file_name" is included in the main C program where "#include <file_name>" is mentioned.

C

```
#include <stdio.h>

#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\?'

void main()
{
  printf("value of height    : %d \n", height );
  printf("value of number : %f \n", number );
  printf("value of letter : %c \n", letter );
```

```
    printf("value of letter_sequence : %s \n", letter_sequence);
    printf("value of backslash_char  : %c \n", backslash_char);

}
```

Output:

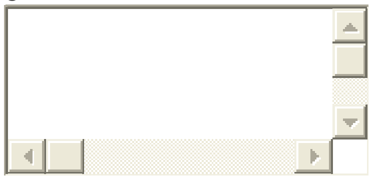
```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

Example program for conditional compilation directives:

a) Example program for #ifdef, #else and #endif in C:

- “#ifdef” directive checks whether particular macro is defined or not. If it is defined, “If” clause statements are included in source file.
- Otherwise, “else” clause statements are included in source file for compilation and execution.

C



```
#include <stdio.h>
#define RAJU 100

int main()
{
   #ifdef RAJU
   printf("RAJU is defined. So, this line will be added in " \
          "this C file\n");
   #else
   printf("RAJU is not defined\n");
   #endif
   return 0;
}
```

Output:

```
RAJU is defined. So, this line will be added in this C file
```

b) Example program for #ifndef and #endif in C:

- #ifndef exactly acts as reverse as #ifdef directive. If particular macro is not defined, “If” clause statements are included in source file.
- Otherwise, else clause statements are included in source file for compilation and execution.

C

```c
#include <stdio.h>
#define RAJU 100
int main()
{
  #ifndef SELVA
  {
    printf("SELVA is not defined. So, now we are going to " \
           "define here\n");
    #define SELVA 300
  }
  #else
  printf("SELVA is already defined in the program”);

  #endif
  return 0;

}
```

Output:

```
SELVA is not defined. So, now we are going to define here
```

c) Example program for #if, #else and #endif in C:

- “If” clause statement is included in source file if given condition is true.
- Otherwise, else clause statement is included in source file for compilation and execution.

C

```c
#include <stdio.h>
#define a 100
int main()
{
  #if (a==100)
  printf("This line will be added in this C file since " \
         "a \= 100\n");
  #else
  printf("This line will be added in this C file since " \
         "a is not equal to 100\n");
  #endif
  return 0;
}
```

Output:

```
This line will be added in this C file since a = 100
```

Example program for undef in C:

This directive undefines existing macro in the program.

C

```c
#include <stdio.h>

#define height 100
void main()
{
  printf("First defined value for height    : %d\n",height);
  #undef height          // undefining variable
  #define height 600     // redefining the same for new value
  printf("value of height after undef \& redefine:%d",height);
}
```

Output:

```
First defined value for height : 100
value of height after undef & redefine : 600
```

Example program for pragma in C:

Pragma is used to call a function before and after main function in a C program.

C

```c
#include <stdio.h>

void function1( );
void function2( );

#pragma startup function1
#pragma exit function2

int main( )
{
  printf ( "\n Now we are in main function" ) ;
  return 0;
}

void function1( )
{
  printf("\nFunction1 is called before main function call");
}

void function2( )
{
    printf ( "\nFunction2 is called just before end of " \
             "main function" ) ;"
}
```

Output:

```
Function1 is called before main function call
Now we are in main function
Function2 is called just before end of main function
```

More on pragma directive in C:

S.no	Pragma command	description
1	#Pragma startup <function_name_1>	This directive executes function named "function_name_1" before
2	#Pragma exit <function_name_2>	This directive executes function named "function_name_2" just before termination of the program.
3	#pragma warn – rvl	If function doesn't return a value, then warnings are suppressed by this directive while compiling.
4	#pragma warn – par	If function doesn't use passed function parameter , then warnings are suppressed
5	#pragma warn – rch	If a non reachable code is written inside a program, such warnings are suppressed by this directive.