C Language Notes

C Playlist Link :-

https://www.youtube.com/playlist?list=PLu0W_9lII9aiXlHcLx-mDH1Qul38wD3aR

Boilerplate Code

```c
#include <stdio.h>

int main()
{

    return 0;
}
```

First C Program

```c
#include <stdio.h>

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

First C Program Explanation

#include <stdio.h> printf() function is defined in *stdio.h*. So, include the file, also called a *header file* to use the function.

int main() The *main()* function is the entry point to a program.

Curly brackets { } indicate the *beginning* and end of a function (also called a code block). The code inside it determines what the function does when executed.

return 0; indicate successful execution of the program.

Semicolon (;) is used to terminate the program.

C Language Notes

Variables In C

A Variable is a name given to a place in a memory in which we store a value. We can first declare the variable and then initialize a value to it like this :-

```
int Number;
Number = 10;
```

Variable Declaration and Initialization can be done together in one line like this :-

```
int Number = 10;
```

Here, *int* means the variable which has the Integer as a value in it. *int* is a data type. We have to declare the data type of variable in C. More about Data Types Later !

Rules For Declaring Variables In C

1. Variable names can contain alphabets, digits and underscores but cannot start with a digit.
2. No whitespaces and commas allowed.
3. No special character other than underscore allowed.
4. Variables are case-sensitive.

Tokens in C

The compiler breaks a program into the smallest possible units and proceeds to the various stages of the compilation, which is called token.

There are 6 types of Tokens in C :-

1. Identifiers –

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

2. Keywords –

These are reserved words , whose meaning is already known by compiler. These words cannot be used as variable names.

C Language Notes

There are 32 Keywords available in C language :

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

3. Constants –

Constants are also called literals. Constants can be any of the Data Type. It is considered best practice to define constants using only upper-case names. Constant's value cannot be change once declared. We use *const* keyword to define a constant in C. For Example:-

```
const int Number = 10;
```

4. Strings – About Strings Later !

5. Operators –

An operator is a symbol used to perform operations in given programming language.

May be some operators missing in the table and they will be specified with the topic with which they will be used.

There are 8 Types of Operators in C :-

- Arithmetic Operators

Operator	***Use***
+	Addition :- Adds the numbers
-	Subtraction :- Subtracts the numbers
*	Multiplication :- Multiply the numbers
/	Division :- Divide the numbers
%	Modulus :- Gives the remainder of any division , cannot be applied on float and sign is same as numerator.

C Language Notes

- Relational Operators

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

- Logical Operators

Operator	***Use***
&&	Both operands are non – zero then condition is true
||	One operand is non zero then condition is true
!	Reverses the correct State

- Assignment Operators

Operator	***Use***
=	Assign Value from Right to left
+=	Add Right to Left and assign result to Left
-=	Subtract Right to Left and assign result to Left
*=	Multiply Right to Left and assign result to Left
/=	Divide Left to Right and assign result to Left

- Increment and Decrement Operators

Operator	***Use***
++	Increment (Add)
--	Decrement (Subtract)

C Language Notes

- Conditional Operator

Operator	***Use***
? :	Conditional Expression

- Bitwise Operators

Operator	***Use***
&	AND : Both operands are non - zero then TRUE
|	OR : One operand is non - zero then TRUE
^	XOR : One operand is non-zero and other is zero then TRUE
~	Binary One's Complement
<<	Binary Left Shift
>>	Binary Right Shift

- Special / Miscellaneous Operators

Operator	***Use***
sizeof()	Return size of variable
&	Return address of variable
*	Pointer to a variable

Operator Precedence

The precedence of operators determines which operator is executed first if there is more than one operator in an expression. Means that priority is given to which operator in an expression if more than one operator is present.

Associativity of an operator is a property that determines how operators of same precedence are grouped in the absence of parentheses.

Here is the Operator Precedence Table In C :

C Language Notes

Category	***Operators***	***Associativity***
Postfix	() [] . -> ++ --	Left to Right
Unary	+ - ! ~ ++ -- (type) * & sizeof	Right to left
Multiplicative	* / %	Left to Right
Additive	+ -	Left to Right
Shift	<< >>	Left to Right
Relational	< <= > >=	Left to Right
Equality	== !=	Left to Right
Bitwise AND	&	Left to Right
Bitwise XOR	^	Left to Right
Bitwise OR	|	Left to Right
Logical AND	&&	Left to Right
Logical OR	||	Left to Right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= > >= < <= &= ^= |=	Right to Left
Comma	,	Left to Right

Note :-

1. No operator is assumed to be present.
   int R = pd → Invalid
   int R = p*d → Valid
2. There is no operator to perform exponentiation in C.
   However , we can use pow (x , y) from <math.h> .

6. Special Symbols –

For Example, Operators

C Language Notes

Data Types In C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

There are 4 Kinds of Data Types in C :-

1. Basic – int , char , float , double

Data Type	***Use***
int	Stores Integer Value in it.
char	Stores a character in it.
float	Stores a Decimal in it.
double	Stores big Decimal in it.

2. Derived – array , pointer , structure , union (Later !)
3. Enumeration – enum (Later !)
4. Void – void means nothing (empty)

Type Conversion in C

An Arithmetic operation between :-
Int and Float results Float
Int and Int results Int
Float and Float results Float

Comments In C

Comments are used to add notes to our program. Comments are not executed and ignored by the compiler.
It is mostly used to specify which line of code performs which function .
There are two types of comments in C :-

1. Single Line Comment –

```
// Single Line Comment
```

2. Multi Line Comment –

```
/* Multi Line
Comment*/
```

C Language Notes

Escape Sequences In C

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character. It is composed of two or more characters starting with backslash (\ ) . For example:- \n represents new line.

Escape sequences	Character represented
\a	Alert (bell, alarm)
\b	Backspace
\f	Form feed (new page)
\n	New-line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\\	Backslash
\nnn	Octal number
\xhh	Hexadecimal number
\0	Null

C Language Notes

Format Specifiers In C

Format Specifier is a way to tell the compiler that what type of data is present in the variable during taking input and displaying output to the user. There are many Format Specifiers in C. All Format Specifiers in brief with the Use:-

Format Specifier	***Use***
%c	Character
%d	Signed Integer
%e or %E	Scientific Notation of Float
%f	Float Values
%g or %G	Same as %e or %E
%hi	Signed Integer (short)
%hu	Unsigned Integer (short)
%i	Unsigned Integer
%l or %ld or %li	Long
%lf	Double
%Lf	Long Double
%lu	Unsigned Int or Unsigned Long
%lli or %lld	Long long
%llu	Unsigned Long long
%o	Octal Representation
%p	Pointer
%s	String
%u	Unsigned Int / Pointer
%x or %X	Hexadecimal
%n	Prints Nothing
%%	Prints % character

Taking Input from User

We use inbuilt function in C to take input from user and that is **scanf.** Syntax of using scanf is as follows :-

C Language Notes

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

Format Specifier

Address Of Operator : This is Very Important !
'a' is the name of variable.

Types of Statements in C

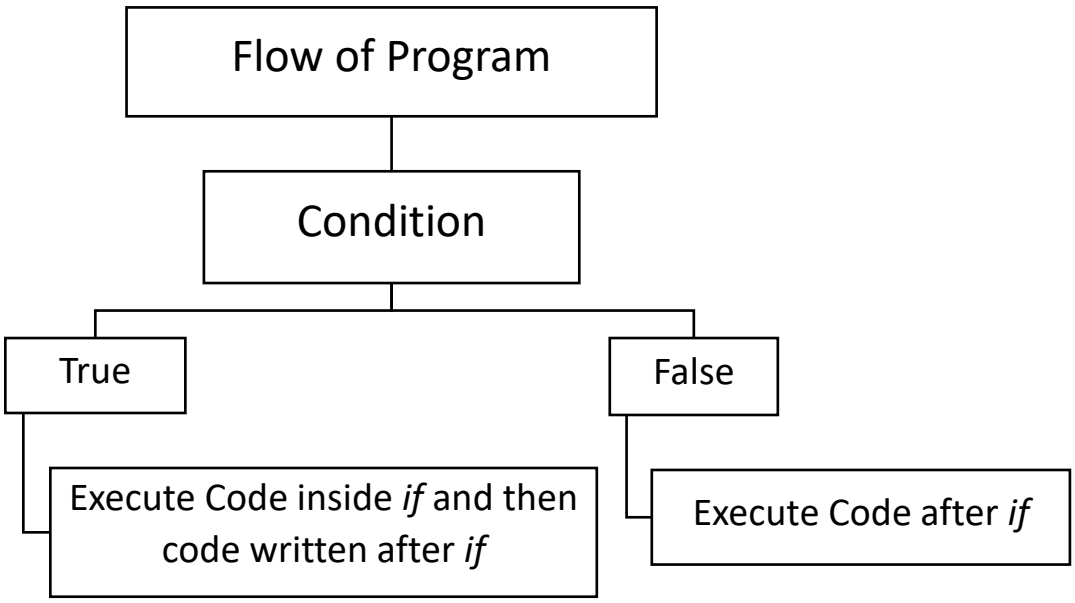
There are 5 types of Statements in C :-

1. Decision Making Statements Or Conditional Statements

These are based on conditions. Conditions are given to computer that if this condition is true then execute this code and so on. There are 4 kinds of Decision-Making Statements –

➢ ' if ' Statement

Syntax – **if (condition) {true}**

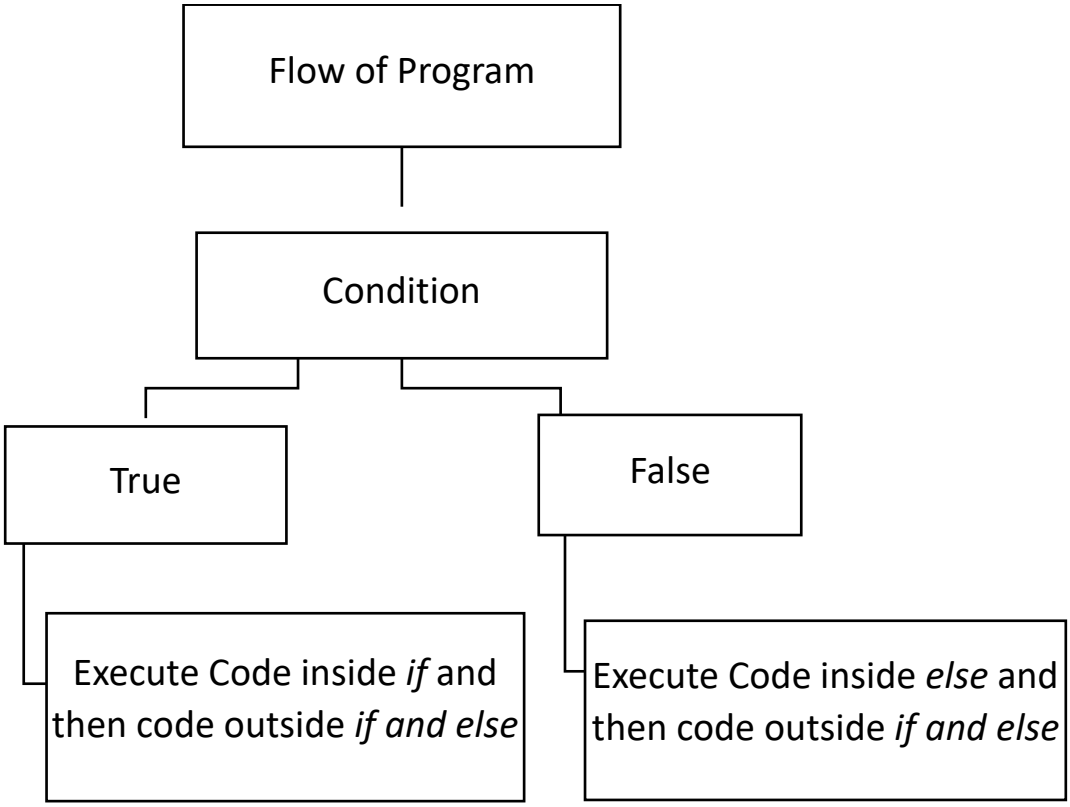



```c
int a = 10;
if (a < 11)
{
    printf("%d is less than 11.\n", a);
}
printf("Done !");
```

C Language Notes

- ‘ if else ’ statement

  Syntax – **if (condition) {true} else { if false }**

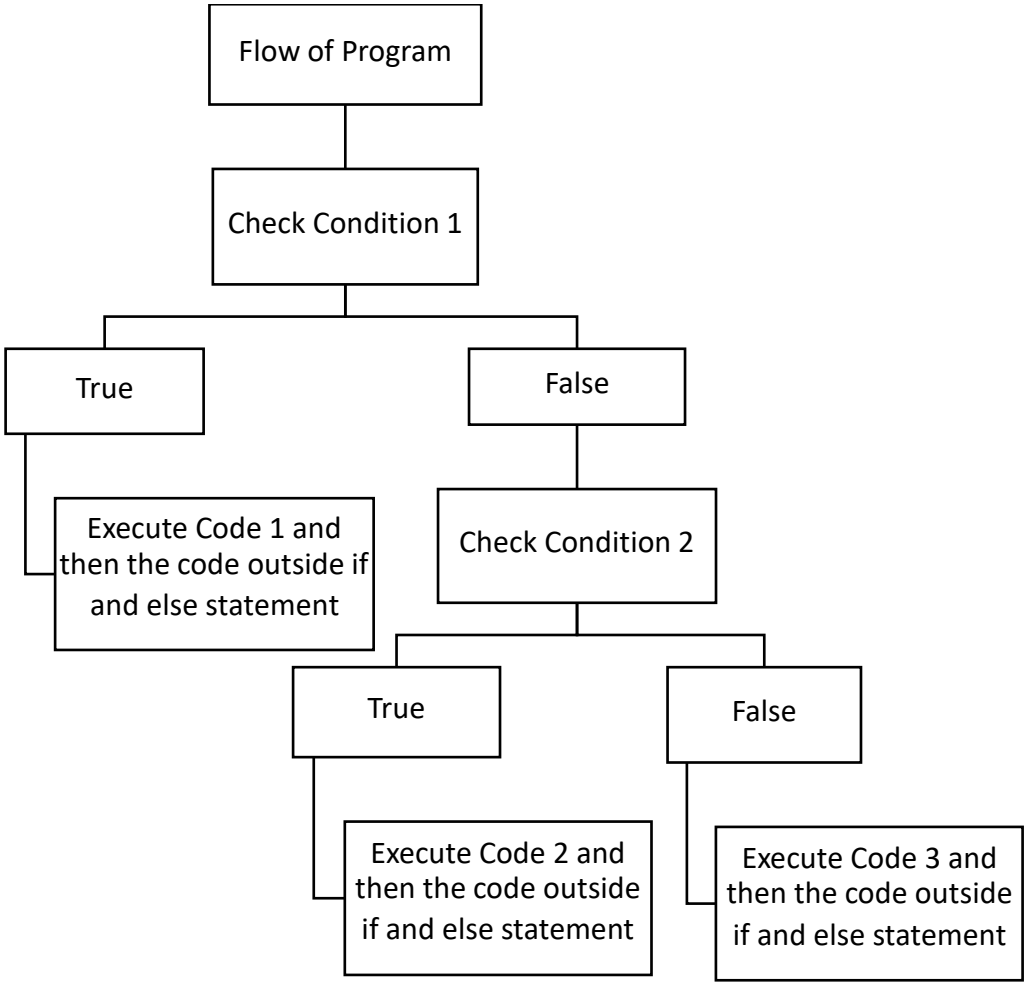



```c
int a = 14;
if (a < 11)
{
    printf("%d is less than 11.\n", a);
}
else
{
    printf("%d is greater than 11.", a);
}
printf("Done !");
```

- ‘ if-else-if ’ ladder

  Syntax – **if (condition 1) {code 1} else if (condition 2) {code 2} ..... else {code 3}**

C Language Notes

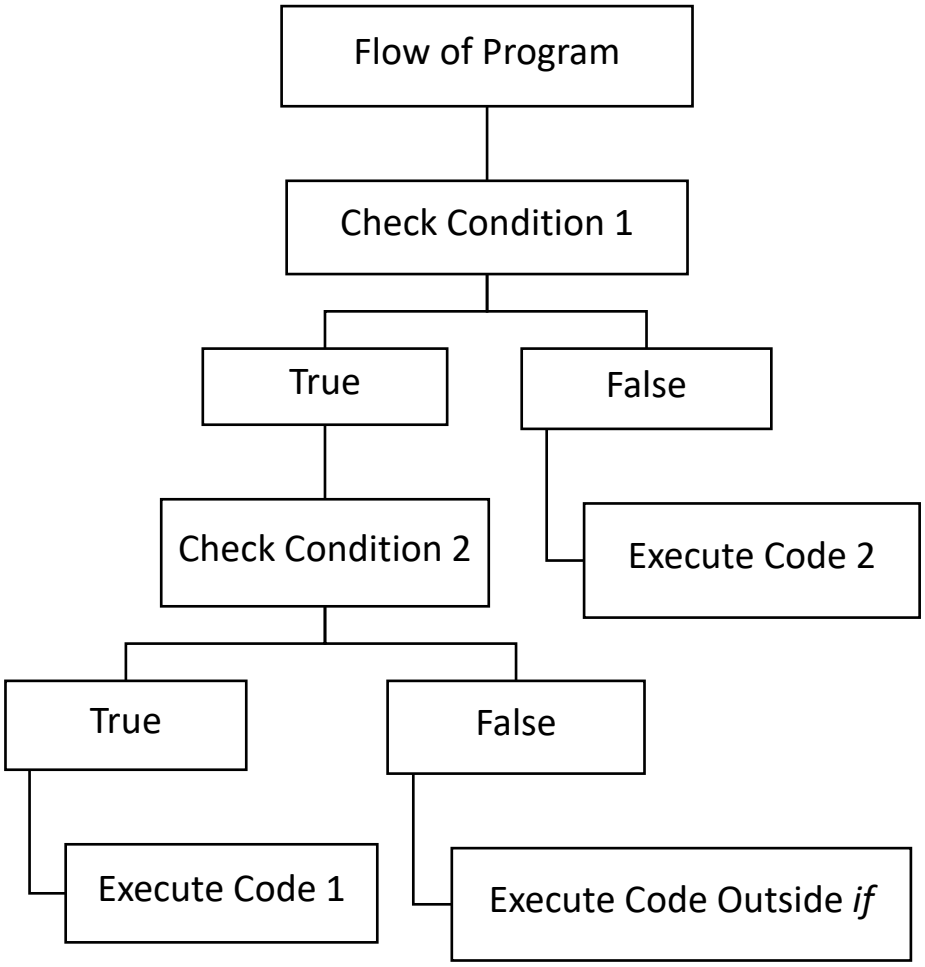



```c
int a = 14;
if (a < 11)
{
printf("%d is less than 11.\n", a);
}
else if (a == 11)
{
printf("%d is equal to 11.\n", a);
}
else
{
printf("%d is greater than 11.", a);
}
printf("Done !");
```

C Language Notes

- ‘ Nested if ’ statement

  Syntax – **if (condition 1) { if (condition 2) {code 1} else {code 2} }**




```c
int a = 14;
if (a > 10)
{
  if (a < 20)
    {
        printf ("Conditions Matched !");
    }
}
else
{
    printf("No Conditions Matched !");
}
```

C Language Notes

Short Hand if - else Statement

Using Conditional Operator –

Syntax – **condition ? code if true : code if false**

2. Selection Statements

Selection Statement is used when we have to make a choice between number of alternatives for a given variable. There is only 1 Selection Statement –

- Switch Case

  Syntax –

  switch (integer or character expression)
  {
  case c1:
  code;
  break;
  case c2:
  code;
  break;
  default :
  code;
  break;
  }

The value of integer or character expression is matched against any of the case ; all the code written after it is executed.

Rules for using switch case

a. Switch expression and value must be int or char.
b. Case must come inside switch.
c. break is not a must.
d. A switch can occur within another.
e. We can write switch case statements in any order of our choice. (Not necessarily ascending or descending)

C Language Notes

```c
int rating;
printf("Enter your Rating from 1 to 3 : \n");
scanf("%d", &rating);
switch (rating)
{
    case 1:
        printf("Rating is 1.");
        break;
    case 2:
        printf("Rating is 2.");
        break;
    case 3:
        printf("Rating is 3.");
        break;
    default:
        printf("Rating out of Range !");
        break;
}
```

3. Iteration/Loop Statements

Loops helps us if we want to execute the piece of code multiple times without repeatedly writing the code. Loops check the condition and if it is true , then keeps executing the code until the condition becomes false.There are 3 Kinds of Loop Statements in C :-

➢ While Loop

Syntax – **while (*condition*) {*code*}**

Note **:-** While loop first checks the condition then executes the code.

C Language Notes

```c
int a = 1;
while (a < 11)
{
  printf("%d\n", a);
  a++;
}
```

➢ Do While Loop

Syntax – **do {*code*} while (*condition*)**

```c
int a = 1;
do {
    printf("%d\n", a);
    a++;
} while (a < 11);
```

Note **:-** Do While loop first executes the code then checks the condition.

➢ For Loop

Syntax – **for (*initialize; test; increment or decrement*) {*code*}**

```c
for (int a = 1; a < 11; a++)
{
  printf("%d\n", a);
}
```

4. Jump Statements

There are 4 Kinds of Jump Statements in C :-

➢ Break Statement

“Break” Statement can be used with loops and switch casw statements. It is used to bring program control out of the loop.

Syntax – **break;**

C Language Notes

```c
int a = 1;
while (a < 11)
{
  printf("%d\n", a);
    a++;
    if (a == 6)
    {
    break;
     }
}
```

➢ Continue Statement

"Continue" Statement is used to bring the program control to the next iteration of loop. It is mainly used for a condition so that we can skip some lines if code for a particular condition. It skips the code written after it and goes to the next iteration of the loop.

Syntax – **continue;**

```c
int a = 1;
while (a < 11)
{
    printf("%d\n", a);
    a++;
    if (a == 6)
    {
        continue;
        printf("Value of A is 6");
    }
}
```

C Language Notes

Label Syntax

Label must be created inside the *main()* function and after and outside the loop. Syntax –

label_name **:**

code

```
end:
    printf("End !");
```

➢ Goto Statement

“Goto” statement is used to transfer program control to a specified label. It is also used when we want to break multiple loops uing a single statement at the same time.

Syntax – **goto *label_name;***

```
for (int i = 1; i < 10; i++){
  printf("%d\n", i);
  while (i == 6)
  {
    goto end;
  }
}
```

Typecasting In C

Typecasting means converting one data type to another.

Syntax for typecasting is as follows :

(data_type) *variable_name*

```
float num;
scanf("%f", &num);
printf("Before Typecasting :- %f\n", num);
printf("After Typecasting :- %d", (int) num);
```

C Language Notes

Pointers In C

A pointer is a variable which stores the address (memory location) of another variable. For Example :-

```
int a = 6;
// Address of a --> 845623
int b = 845623;
// Address of b --> 845634
// Here b is a pointer as it is storing the address of variable a
```

The ‘ address of ’ operator (&) is used to obtain the address of given variable. If you refer the examples above :-

&a → 845623

&b → 845634

Format specifier for printing pointer address is *“%u”*.

The *value at address* operator (*) is used to obtain the value of present at a given memory address.

We can declare a pointer like this :

```
int *b = &a;
```

Pointer can be type of int , char and float as well.

We can create pointer of a pointer as well. By increasing number of *value at address* operator (*) Like this :-

```
int **c = &b;
```

This code shows that how value of variable is obtained using a pointer by just using *value at address* operator while passing the pointer as an argument. (In 4th Line)

```
int a = 23;
int *ptr = &a;
printf("The Address of variable is %u\n", ptr);
printf("The Value of variable using ptr is %u", *ptr);
```

C Language Notes

There are some kinds of pointers which can be used :

1. **Void Pointer** is a pointer that has no data type associated with it. It can be easily type casted to any pointer type. C does not allow void pointers to be dereferenced, meaning that we have to typecast the pointer every time it is being used. We cannot use pointer arithmetic with void pointers. Its use is that it is a general-purpose pointer.

```c
char a = 'P';
int b = 10;
void *ptr = &a; // ptr stores address of int a
ptr = &b; // ptr now stores address of char b

printf("%d\n", (int *)ptr); // Will output the value
of ptr
printf("%d", *(int *)ptr); // Will output the value of
the variable whose address is stored in ptr
```

2. **NULL Pointer** is a pointer that does not point to any memory location. It generally points to NULL or 0th memory location, so in simple words, no memory is allocated to a NULL pointer. A null pointer is guaranteed to compare unequal to any pointer that points to a valid object. A Null pointer cannot be dereferenced in C (a pointer it expects to be valid, but is NULL, typically causing a crash or exit). This gives programmer a functionality to check whether the pointer is legitimate or not. It is used to initialize a pointer variable if we don't want to assign it a valid address for some reason. And to pass a Null pointer to a function argument when we don't want to pass any valid memory address. An uninitialized pointer has a garbage value and Null pointer has no value. Most of the compilers determine Null as 0. We use it for error handling in C Programming.

C Language Notes

```
int a = 10;
int *ptr = NULL;
printf("The Address Of Variable is %u.", *ptr); //
Program will crash because of this line
```

3. **Dangling Pointer** is a pointer that is pointing to a memory location that has been freed or deleted. Dangling pointers arise during object destruction, when an object with an incoming reference is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory. The system may reallocate the previously deleted memory; the unpredicted result may occur as the memory may now contain different data.
   There are some causes of dangling pointers :

1. Deallocation Of Memory
2. Returning Local Variables In Function Calls
3. Variable Going Out Of Scope

   The dangling pointer introduces nasty bugs in our C programming and these bugs frequently become security holes at a time. These dandling pointer errors can be avoided by initializing the pointer value to the NULL. If we assign the NULL value to the pointer, then the pointer will not point to the memory location that has been freed. By assigning the NULL value to the pointer means that the pointer is not pointing to any memory location.

```
int *ptr = (int *) malloc(sizeof(int));
int a = 10;
ptr = &a;
free(ptr); // Now ptr is a Dangling Pointer
ptr = NULL; // Now ptr is not a Dangling Pointer
```

4. **Wild Pointer** is a pointer that are uninitialized and they point to some arbitrary location in the memory of may cause program to crash and dereferencing these pointers can cause nasty bugs and it is suggested to initialize unused pointers to NULL.

```
int a;
int *ptr; // This is a wild pointer that is storing a
garbage value which may or may not be pointing to a
valid memory location
ptr = &a; // It is no longer a wild pointer
```

Pointer Arithmetic In C

A pointer can be incremented to point to the next memory location of that type. Which means that if *int* takes 4 bytes in a particular architecture, then incrementing or decrementing any value from the pointer variable will add or subtract 4 from it, not the exact value. In short, it increments or decrements the *sizeof (data_type)*.

There are 4 Arithmetic Operations that can be done with pointers :

++ , -- , + , -

```
int a = 7;
int *pointer = &a;
pointer++;
pointer--;
pointer = pointer + 1;
pointer = pointer - 1;
```

C Language Notes

Functions In C

A function is a block of code which performs a particular task. A function can be reused by the programmer in a given program any number of times. We have to take three steps to work with the function and the three ways are :-

Function Prototype is a way to tell the compiler about the function we are going to define in the program. We must add a datatype while declaring the function. Function declaration must be done before *int main* function execution starts.

```
void display();
```

Function Definition contains the exact set of instructions which are executed during the function call. When a function is called from main() ; the main function falls asleep and gets temporarily suspended. During this time , the control goes to the function being called. When the function body is done, the main resumes. The Function Definition must be done after the *return* is called.

```
void display() {
  printf("Prachi");
}
```

Function Call is a way to tell the compiler to execute the function body (Function Definition) at the time the call is made. Function call must be done inside the int main function.

```
display();
```

If display(); function is called , it will print *Prachi* on screen !

Passing Values to Functions In C

We can pass values to a function and can get a value in return from a function.

C Language Notes

Parameters are the values or variable placeholders in function definition.
Arguments are the actual values passed when the function call is made.
A function can return only 1 value at a time. If the value of variable passed during the function call is changed in the function definition, then it will not affect the actual variable passed because C language passes the copy of variable to a function.

Types Of Functions In C

There are 2 types of function in C :-

1. **Library Functions –**
   Functions we use from the header files.
2. **User Defined Functions –**
   These are the functions declared and defined by the user.

Types Of Function Calls In C

Based on how we pass arguments to the function, function calls are of 2 Types in C :

1. **Call by Value –** The values of the arguments are passed to the function.
2. **Call by Reference –** The address of the variable is passed to the function as arguments.

Recursion In C

A function in C can call itself. When a function calls itself , It is known as Recursion. Recursion is sometimes the most direct way to code an algorithm. The condition which doesn't call the function any further in a recursive function is called as the base condition. Sometimes , due to a mistake made by the programmer , a recursive function can keep running without returning anything resulting in a memory error.

C Language Notes

```
int factorial(int number)
{
    if (number >= 1)
        return number * factorial(number - 1);
    else
        return 1;
}
```

Here , factorial calls itself till the number reaches 1. And when a function call itself , it is called Recursion.

Arrays In C

An array is a collection of similar elements. One variable is Capable of storing multiple values. Syntax for defining an array is as follows :

```
int array [2];
```

We have to keep this thing in mind that in C language, Array Index starts with 0. We can now assign values to *Marks* array like this :

```
array[0] = 90;
array[1] = 95;
array[2] = 100;
```

We can access these values like this :

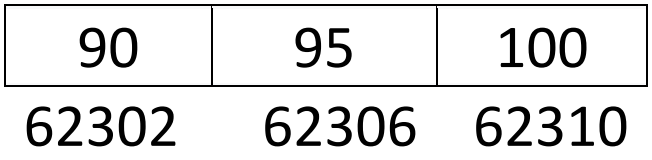
```
printf("%d", array[2]);
```

We can do declaration and initialization of an array together like we used to do with variables like this :

```
int array [3] = {90, 95, 100};
```

The code example above will reserve 4x3=12 bytes in memory. 4 bytes for each integer.

C Language Notes



(Memory Address Of The Integers)

Multidimensional Arrays can be used in C language. They can be 2-dimensional, 3-dimensional, n-dimensional. Syntax is as follows:

Data_type array_name = [number_of_dimensions] [number_of_values_in_each_dimension]

```
int array [2][3] = {
{1, 2, 3, 4},
    {5, 6, 7, 8}
};
```

Arrays can be passed as function arguments likewise the way call by value is passed.

Strings In C

A string is a 1-d character array terminated by a null(‘ \0 ’). The null character is used to denote string termination, characters are stored in contiguous memory locations.

Since string is an array of characters, it can be initialized and printed as follows:

```
char character[] = "Prachi";
printf("%s", character);
```

A string is sorted just like an array in the memory and is stored in contiguous block of memory.

We can take string as input like this :

C Language Notes

```
char input[6];
scanf("%s", &input);
```

gets() is a function that can be used to receive a multi-word string.

Multiple *gets()* calls will be needed for multiple strings.

puts() is a function that prints the string and places the cursor on the next line.

```
char string[6];
gets(string);
puts(string);
```

Strings can also be declared using pointers like this :

```
char *pointer = "Prachi";
```

This tells the compiler to store the string in the memory and the assigned address is stored in a char pointer.

Once the string is defined, it cannot be initialized to something else but a string defined using pointers can be reinitialized.

Here are some functions that can be used on strings which comes under *<string.h>* header file:

strlen() function is used to count the number of characters in the string excluding the null ('\0') character.

strcpy() function is used to copy the content of second string (target) into first string (source) passed to it.

C Language Notes

Target string should have enough capacity to store the source string.

strcat() function is used to concatenate two strings. It will not leave any space in between while concatenating the strings if not given between the two strings while initializing.

strcmp() function is used to compare two strings. It returns 0 if strings are equal and Negative value if first strings mismatching character's ASCII value is not greater than second string's corresponding mismatching character. It returns positive values otherwise.

Structures In C

As arrays and strings can hold similar type of data (like an array can be either integer or float or something else), structures can hold dissimilar data. Structures are user-defined data type. One important note that structures are defined before the main function like wise we used to do in Function Prototype. Structures can be created using *struct* keyword as follows:

```
struct Employee
{
    int Mobile_Number;
    float salary;
    char Name[10];
};
```

All the variables declared inside structures are known as *members of structure*.

C Language Notes

A structure in c is a collection of variables of different types under a single name. We can create as many *Employees* as we want with their Name, Salary and Mobile Number (according to the above example). There are 2 or more valid syntax for doing this but one of the common syntaxes (according to the above example) which will be used inside the *main* function is as follows:

```
struct Employee e1;
```

Here *e1, e2, e3* are employees which will contain all the different information as Name, Salary and Mobile Number.

Values can be assigned and accessed for *e1, e2, e3* in 2-3 ways but one of the most common ways is like this:

```
e1.salary = 100000;
printf("%f", e1.salary);
```

Just like an array of integers, an array of floats, and an array of characters, we can create an array of structures. Here also There are 2 or more valid syntax for doing this but one of the common syntaxes (according to the above example) which will be used inside the *main* function is as follows:

```
struct Employee Juniors[100];
```

Here an array named *Juniors* is created of type *Employee* and it will work same as arrays and can be initialized and accessed using the *structure member operator (.)* which is also known as *dot operator* like this:

C Language Notes

```
Juniors[0].salary = 100000;
printf("%f", Juniors[0].salary);
```

Structures are stored in contiguous memory locations. In an array of structures, these employee instances are stored adjacent to each other. These members are stored in different memory locations which means that the size of structure will be equal to the addition of size of its members.

Pointers can be also used with Structures as follows :

```
struct Employee *ptr;
ptr = &e1;
```

And the data can be also initialized and accessed using pointers like this :

```
(*ptr).salary = 100000;
printf("%f", (*ptr).salary);
```

An alternative way is to do it like this :

```
ptr -> salary = 100000;
```

In the code above, the *arrow operator (->)* is used.

Structure can be passed to function while declaring function prototype like this :

```
void show(struct Employee e1);
```

Typedef Keyword In C

It is a keyword that is used to assign alternative names to existing datatypes. We use typedef with user defined datatypes, when names of the datatypes become slightly

complicated to use in programs. And then we can use that alias name at the place of previous name. Syntax is like this :
typedef previous_name alias_name;

```
typedef unsigned long ul;
ul a, b, c;
```

typedef can be used with pointers also like this :

```
typedef int *intPointer;
intPointer a;
```

We can use the alias name *intPointer* declared above like :

```
int b = 14;
a = &b;
```

Main use of typedef is with structures like this and one note that it should be done before the *main()* function starts:

```
typedef struct Student
{
    int Id;
    char Name[30];
} Std;
```

Here, *Std* is the alias name of *struct Student* which can be used like this inside the *main()* function:

```
Std s1;
s1.Id = 1;
```

Unions In C

Just like Structures, the union is a user-defined data type. All the members in union share the same memory location. One of the advantages of using a union is that it provides an efficient way of reusing the memory location, as only one of its members can be accessed at a time. Union is especially used when the memory is limited and each byte is significant

and when memory should be reused efficiently. A union is declared and used in the same way we declare and use a structure but we use *union* keyword for creating a union as follows:

```
union Student
{
    int Id;
    int Marks;
    char Favourite_Character;
    char Name[20];
};
```

In the above example, there are four members in the above union but we can use any one of them at a time because only one location is allocated for all the members, irrespective of their size. The size of above union will be 20 as the largest member of the union is *Name* with size 20. And all the members of union Student will be stored in just 20 bytes which means they will share the same memory location.

We can access these members just like we used to do with the structures :

```
union Student s1;
s1.Marks = 100;
s1.Id  = 1;
s1.Favourite_Character = 'K';
strcpy(s1.Name, "Prachi");
```

If we print all these values after initializing the values, the value which will be initialized at last (in this example it is

s1.Name) will be printed correctly and all other values will be garbage values. As we can only use one member at a time.

Just like structures, we can create an array of union also with the same syntax just the keyword will be *union* not struct which should be done inside *main()* function like this :

```
union Student Coder[100];
```

Same way we can access the elements of array of union like this :

```
Coder[0].Id = 1;
printf("%d", Coder[0].Id);
```

Pointers can be also used with Unions as follows :

```
union Student *ptr;
ptr = &s1;
```

And the data can be also initialized and accessed using pointers like this :

```
(*ptr).Marks = 90;
printf("%d", (*ptr).Marks);
```

An alternative way is to do it like this :

```
ptr -> Marks = 90;
```

In the code above, the *arrow operator (->)* is used.

Union can be passed to function while declaring function prototype like this :

```
void show(union Student e1);
```

C Language Notes

Local And Global Variables In C

Scope refers to the variable's availability at places.

Local variables are declared inside a function or a block of code; they cannot be accessed outside the function. The local variables can be used only by statements that are inside that function or block of code. The system does not initialize local variables, we must initialize it ourself. The scope of these variables will be within the function only.

Global Variables are defined outside a function, usually in the main. Global Variables hold their values, and we can access them inside any of the functions defined for the program. Global variables are initialized by the system automatically when we define them.

If the local and global variables have the same name. the local variable will take preference.

Formal And Actual Parameters In C

Actual Parameters are values that are passed to the called function from the main function.

Formal Parameters are variables declared in the function prototype or definition.

When a method is called, the formal parameter is temporarily "bound" to the actual parameter.

C Language Notes

Static Variables In C

A static variable is known to retain the value even after they exit the scope. Static variables retain their value and are not initialized again in the new scope. The static variable until the end of the program is kept in the memory, whereas a normal variable is destroyed when a function is over. They can be defined inside or outside the function. Static variables are local to the block. The default value of static variables is zero. The keyword *static* is used to declare a static variable like this:

static datatype variable_name = variable_value

```
static int var = 14;
```

Static Global Variables are declared with a static keyword outside the function. This variable will be accessible throughout the program.

Static Local Variables are declared with a static keyword inside a function. The scope of the static local variable will be the same as the local variables, but its memory will be available throughout the execution of the program.

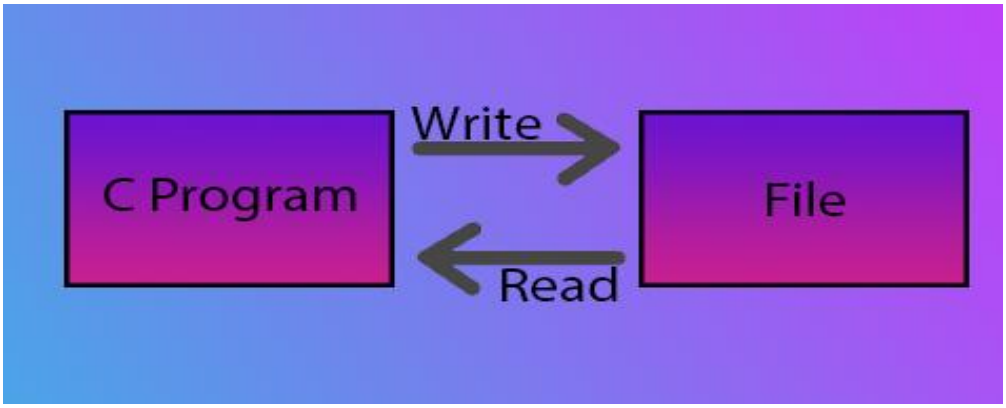
Properties of Static Variable:

1. Static Variable Memory is allocated within Static Variable.
2. A static variable will retain the value even after they exit the scope.
3. Static variable memory is available throughout the program.
4. If we do not assign any value to the static variable, then the default value will be 0.

C Language Notes

File I/O In C

A random-access memory is volatile memory and its content is lost once the program terminates. Files are used to store content hence reducing the program's size. A file data stored in a storage device. A C program can talk to the file by reading content from it and writing content to it. Files are stored in non – volatile memory.



First, we have to understand difference between Volatile and Non – Volatile Memory :

In **Volatile Memory,** The data can only remain while the computer's power is on. It can only hold information when having a constant power supply. And it will hold data for a short period.

In **Non – Volatile Memory,** The data can also remain while the computer's power is off. It can also hold information, in case of inconstant power supply. And it will hold data for a long term.

The File is a structure that needs to be created for opening the file. A file pointer is a pointer to this structure of the file.

C Language Notes

File pointer is needed for communication between the file and the program.

A File pointer can be created like this :

```
FILE *ptr = NULL;
```

It is a good practice if we initialize the file pointer as NULL in the beginning.

There are some modes in C in which a file can be opened and used :

r opens a file in read mode

w opens or create a text file in write mode

a opens a file in append mode

a+ appends in a file

r+ opens a file in read and write mode but does not delete the content inside the file if it is present and writes in the file if commanded

w+ opens a file in the write mode and deletes the content inside the file if it is present and writes given content in the file

C provides a number of build-in function to perform basic file operations:

fopen() function creates a new file and opens it or opens an existing file

C Language Notes

```
ptr = fopen("file.txt", "r");
```

fclose() function closes a file

Files does not automatically get closed after working with them in C. We have to close them manually.

```
fclose(ptr);
```

fgetc() function reads a character from a file and it takes only one argument that is file pointer.

```
char ch = fgetc(ptr);
printf("%c", ch);
```

fputc() function writes a character to a file and it takes two arguments first is the character to be written and second is file pointer.

```
ptr = fopen("file.txt", "w");
fputc('S', ptr);
```

fscanf() function reads a set of data from a file and it takes three arguments first is file pointer, second is format specifier and third is variable name

```
ptr = fopen("file.txt", "r");
char str[10] = "Hello";
fscanf(ptr, "%s", str);
```

fprintf() function writes a set of data to a file and it takes three arguments first is file pointer, second is format specifier and third is variable name

C Language Notes

```
ptr = fopen("file.txt", "w");
char str[10] = "Hello";
fprintf(ptr, "%s", str);
```

fputs() function writes a string in the file and it takes two arguments first is string to be written and second is file pointer

```
ptr = fopen("file.txt", "r+");
fputs("Prachi", ptr);
fclose(ptr);
```

fgets() function reads a string from a file and it takes three arguments first is variable name, second is size of string we want to read and third is file pointer

```
ptr = fopen("file.txt", "r");
char str[10];
fgets(str, 7, ptr);
```

Many other functions are also available in C which can be used to work with files but these are the major ones and other functions can be searched on Internet.

Dynamic Memory Allocation In C

Dynamic memory allocation is the process of allocation of memory space at the run time. We use this concept to reduce the wastage of memory.

Memory allocation in C can be divided into four segments :

1. **Code** composes of all the text segment of our program.

C Language Notes

2. **Variables** composes of both static and global variables and it is further divided into two segments, depending upon the data they store**.**

   a) **Data Segment** stores initialized data, i.e., whose value is already given.
   b) **BSS Segment** stores uninitialized data, i.e., whose value is not given.

3. **Stack** is a LIFO (Last In First Out) data structure. It's size increases when the program moves forward. It basically stores functions and its data inside it and the data will be destroyed when the function returns. Initially, the stack looks like a bucket in which the last entry to be inserted will be the first one to get out.

If we push a function A into the stack. Function A will start executing. Now the function A is calling another Function B during its execution. The Function B will be pushed into the stack, and the program will start executing B. Now, if B is calling another function C, then the program will push C into the stack and will start its execution. Now, after C has been executed completely, the program will pop C from the stack as it was the last one in and start executing B. When B has been executed completely, it will be popped out, and A will start executing until the stack becomes empty.

4. **Heap** is a tree-based data structure. It's size increases when we allocate memory dynamically. To use the heap data structure, we have to create a pointer in our main function that will point to some memory block in a heap. The disadvantage of using heap is that the memory will not get freed automatically when the pointer gets overwritten.

C Language Notes

Stack Overflow is a condition when a stack gets exhausted due to bad programming skills or some logical error.

In **Static Memory** allocation is during compile time, is a non – reusable memory and is a less optimal way.

In **Dynamic Memory** allocation is at run time or execution time, is a reusable memory and is a more optimal way.

We have four functions for allocation of memory using heap and these functions are in *<stdlib.h>* header file so we must include it :

1. **malloc()** stands for memory allocation. It reserves a block of memory with the given number of bytes. The return value is a void pointer to the allocated space. Therefore, the void pointer needs to be type casted to the appropriate type as per the requirements. However, if the space is insufficient, allocation of memory fails and it returns a NULL pointer. All the values at allocated memory are initialized to garbage values. Syntax for using *malloc()* :

   ptr = (ptr_type*) malloc (size_in_bytes)

```
int *ptr;
ptr = (int *)malloc(2 * sizeof(int));
// Using sizeof operator here because size of data
type may differ from system to system
// Values can be assigned to 2 integers like this in
the format of array
ptr[0] = 6;
ptr[1] = 10;
```

C Language Notes

2. **calloc()** stands for contiguous allocation. It reserves *number* of blocks of memory with the given number of bytes. The return value is a void pointer to the allocated space. Therefore, the void pointer needs to be type casted to the appropriate type as per the requirements. However, if the space is insufficient, allocation of memory fails and it returns a NULL pointer. All the values at allocated memory are initialized to 0. Same as *malloc* we can initialize values using array format. Syntax for using *calloc()* is as follows :

   ptr = (ptr_type*) calloc (number_of_blocks_of_memory, size_in_bytes)

```
int *ptr;
ptr = (int*) calloc(10 , sizeof(int));
```

3. **realloc()** stands for reallocation. It is used in cases where the dynamic memory is insufficient or wants to increase the already allocated memory to store more data. Its syntax is simple as we just have to overwrite the memory already allocated as a parameter in the function while providing the data related to the pointer. Syntax for using *realloc()* is as follows :

   ptr = (ptr_type*) realloc (ptr, new_size_in_bytes)

```
int *ptr;
ptr = (int*) realloc(ptr, 7*sizeof(int));
```

4. **free()** is used to free up the space occupied by the allocated memory. Syntax for using *free()* is as follows :

   free (ptr)

```
free(ptr);
```

Storage Classes In C

A storage class defines scope, default initial value, and a lifetime of a variable.

Here, scope refers to the variable's availability at places. The initial default value refers to the value present in the variables as default before being initialized, and the lifetime refers to the variable's duration of life.

There are four types of storage classes depending upon the type of variables they store :

1. **Automatic Variables Or auto Storage Class :**

Variables being formed in a function and whose storage class has not been defined initially fall in this category automatically. Its scope is minimum as it can only be accessed inside the function it is initialized in. No other function can access it. Until the variable has been assigned some value, it stores garbage value as default. Their lifetime depends upon the function block's length as the lifetime is until the function block's end. It is declared using *auto* keyword.

```
auto int sum;
int sum;
// Both Are Same
```

C Language Notes

2. **External Variables Or extern Storage Class**

These sorts of variables are defined outside the function, hence can be used inside any function, meaning that they can be used globally. Their initial value is set to 0. As they can be used throughout the program, so their lifetime equals the lifetime of the program. Too many global variables in a program can cause security issues and also are not usually recommended. It is declared using *extern* keyword. Using the extern keyword, we inform our compiler that the variable is already declared at some other place. By doing so, we can use the same variable with the same space, without allocating its new memory and accessing the same variable in some other file. But we must include that file first. it will automatically access it from the other file.

main.c :

```
#include "temp.c"
extern int a;
```

temp.c :

```
#include <stdio.h>
int a = 60;
```

3. **Static Variables Or static Storage Class**

Already learnt about it but still a recap and extra notes :
Static variables are a little bit technical as their lifetime is throughout the program, but their scope is limited to the function they are initialized in. It comes in handy when we are changing their value in the program as the program will store the new value, overwriting the previous one. Their initial default value is 0.

```
static int num = 14;
```

4. **Register Variables Or register Storage Class**
Its scope is limited to the function it is defined in, the initial default value is 0, and lifetime is till the end of the function block. Now the major difference between it and the others is that it requests the CPU's register memory instead of the local memory for fast access. It is usually used for the programs that need to be accessed faster than the other.

```
register int num = 14;
```

Pre-processor Commands In C

There are four phases for the C program to become executable :

1. **Pre-processing** includes removing of comments, expansion of macro, and expansion on include files.
2. **Compilation** includes generation of assembly level instructions.
3. **Assembly** includes conversion of assembly level instructions to machine code.
4. **Linking** includes resolving of function calls means it links the function implementation.

The C Pre-Processor is not a part if the compiler. The C Pre-Processor is just a text substitution tool, and it instructs the compiler to do required pre-processing before the actual compilation process. We refer the C Pre-Processor as CPP.

Pre-Processor directives like *#include* come into effect as soon as they are seen and keep working until the end of the

file that contains them; the program's block structure is irrelevant. In C , Pre-Processing directives are lines in the program that start with the hash symbol (#). This hash symbol is followed by an identifier that is the directive name. For example , *#include.* Whitespace is not allowed before or after the hash symbol. The Pre-Processor in C does not know about the scope rules of C Language. Here are some important C Pre-Processor directives :

Directives	**Use**
#define	Will substitute a pre-processor macro.
#include	Insert a particular header from another file.
#undef	Can undefine a pre-processor macro.
#ifdef	Will return true if macro is defined.
#ifndef	Will return true if macro is undefined.
#if	Will test if a compile time condition is true.
#else	Will run when compile time condition is false
#elif	Can define #else and #if in one statement
#endif	Will end pre-processor conditional
#error	Can print the error message on stderr.
#pragma	issue the special commands to the compiler, using a standardized method.

C Language Notes

Pre-Defined Macros In C

A Pre-Defined macro is a macro that has already been defined or understood by C pr-processor and foes not need a definition. Here are some Pre-defined Macros In C :

Macros	Use
__DATE__	Prints the current date on to the screen. The date format it follows is MMMDDYYY.
__TIME__	Prints the current time on to the screen. The date format it follows is HH:MM:SS.
__FILE__	Prints the current file name on to the screen. The name will be printed as a string literal.
__LINE__	Prints the current line number on to the screen. The number will be printed as a decimal constant.
__STDC__	Used to check whether our program is being compiled using ANSI standard or not. It will return 1 if true.

Command Line Arguments In C

If we need to pass arguments from the command line to the program a set of inputs. They are used to supply parameters to the program when it is invoked. It is mostly used when you need to control your program from the console.

There are 2 parameters in C which we can use to pass command line arguments to the program/function :

1. **argv :** The *arg* stands for argument and *v* stands for vector. Vector can be said as *one-dimensional array* in this case. *argv* stores the pointer to the arguments passed to the program in an array of strings.

2. **argc :** The *arg* stands for argument and *c* stands for count. It stores the total number of arguments passed to the program/function. The first argument count is reserved for the executable program's name, and next ones are for the arguments passed to the program/function.

Syntax/Example of a command line utility is as follows :

```
int main(int argc, char const *argv[])
{
    for (int i = 0; i < argc; i++)
    {
        printf("This argument number at index %d have
the value of %s\n", i , argv[i]);
    }
}
```

Now this can be accessed through the command line by typing this in the terminal :

File_name.exe arg1 arg2….

Example :

```
demo.exe Prachi loves Programming.
```

Function Pointers In C

We can have pointers pointing to functions as well and they are useful to implement *callback functions*. They point to a specific code block not value/data. Syntax is as follows :

Function_Data_Type (*Name_Of_Pointer) (Data_Type_Of_Function_Arguments)

C Language Notes

```
void (*ptr)(int, int);
ptr = &Function;
```

Declaration and Initialization can be done in one statement like this :

```
void (*ptr)(int, int) = &Function;
```

Callback Functions In C

Function pointers are used to pass a function to a function as an argument. This passed function can then be called again. That is why it is known as callback function. Syntax is as follows :

```
Function(ptr); // After Declaring and Initializing ptr
```

Enumeration In C

Enumeration or Enum in C is a special kind of data type defined by the user. It consists of constant integrals or integers that are given names by a user. The use of enum in C to name the integer values makes the entire program easy to learn, understand, and maintain by the same or even different programmer. It is done by using *enum* keyword. Syntax is as follows :

enum enum_name{int_const1, int_const2, int_const3, …. int_constN};

```
enum First{Mon, Tue, Wed, Thurs};
```

Here, default value of constants is 0, 1, 2, 3 respectively but they can be changed like this :

C Language Notes

```
enum First{Mon = 4, Tue = 5, Wed = 6, Thurs = 7};
```

They can be accessed like this :

```
enum First day; // Here day is a variable of First enum
day = Wed;
printf("%d", Wed);
```

End Of C Notes !!!