**C Programming**

**Introduction**

* This is the mother language of all the other languages like C++ , JAVA , Python .
* C is a **structure oriented language** unlike C++ , C# and JAVA .

**Reasons of studying C in the age of OOP languages**

* C is faster
* Major parts of Operating systems like Windows , UNIX , Linux and Android are written in C .
* All the device driver programs are exclusively written in C
* Embedded programs in Consumer devices like microwaves , Ovens are written in C .
* Popular gaming Frameworks like DirectX are written in C .
* C provides various language elements that make one to have very close interaction with hardware devices .
* It provides good base for learning other modern languages like C++ , C# , JAVA etc.

**Specialities :-**

* Time and memory efficient.
* Can Interact with hardwares very closely .

**Important terms and definitions : -**

* Keywords -> words which carry special/fixed meaning . Its meaning is already been explained to the compiler/computer .
* There are 32 keywords available in C .
* Semicolon ‘;’ acts as the statement terminator .
* In main() function returning 0 means success .
* **#include** is a **preprocessor directive**
* %d , %c , %fare the **format specifiers**
* **Compiler** converts the the high level language to **machine language**
* **&(ampersand)** is ‘ Address of ’ operator . It gives location number(address) used by the variable in memory
* Variables are the **names given to memory locations .**
* Every compiler is **Platform specific** .
* A platform is a combination of specific OS and microprocessor(i.e **OS + Micropreocessor = Platform**)

**C Instructions**

* **Program** is a set of instructions in order to achieve some certain tasks .
* Types of instructions : -
  + Type declaration instructions – used to declare the type of variables .

ex:- int a , float b etc

* + Arithmetic Instruction – used to perform arithmetic instructions on constants and variables .

s = a + b + 32 , t = 32 \* / a etc

* + Control Instruction – used to control the sequence of execution of various statements in a program
* Arithmetic operators are ‘+’ , ‘-’ , ‘\*’ , ‘/’
* Modulus operator(%) is not used on float
* Using % , sign of the remainder is always same as the sign of numerator
* **ASCII codes** are used to represent any character in the memory , ex:- ASCII code for ‘a’ and ‘b’ are 01100001 and 01100010 respectively
* **ASCII value**s are the decimal equivalent of the **ASCII codes ,** ex:- ASCII values of ‘a’ and ‘b’ are 97 and 98 respectively .
* **‘b’ + ‘a’** , here , addition is performed on the ASCII values of the characters i.e , **97+98**
* **pow(a,b)** is a standard **library function** declared in **<math.h>** header file , used for exponentiation (i.e a raised to power b) .
* **pow()** only works with real numbers .
* **sqrt()** function is used for getting square-root.
* **#include** is a preprocessor directive
* In an operation between **real and integer** always yield **real** . To achieve this integer is first promoted to real .
* **Hierarchy of operators** (priority / precedence):-
  + Parantheses ‘()’
  + \* / %
  + + -
  + =
* **Priority can be changed using ()**
* **Associativity of Operators**
  + When an expression contains two operators of equal priorioty/precedence , the tie between them is settled using associativity of operators
  + \* / follows **left to right** associativity , ex:- 3/2\*5 , / is in left of \* , hence / will be processed first the \*
  + = follows **right to left** associativity , ex:- a=b=2 , rightmost ‘=’ will be performed first and then left one , i.e **at first b = 2 , then a = b**
* **Control Instruction**
  + Specify the order in which the various instructions are to be executed in a program .
  + Types:-
    1. Sequence Control Instruction – It ensures that the instructions are executed in the same order in which they appear in the program
    2. Decision Control Instruction – Takes the decision as to which instruction is to be executed next
    3. Repetition or Loop Control – helps to execute group of statements repeatedly
    4. Case Control Instruction – It allows us to perform instructions as per our choice . OR , It executes only specific **case statements** based on the **switch expression** .

**Decision Control(Control flow/Conditional Statements)**

- **if-else** and **switch-case** statement are the two main decision control instructions

if(i==5) **;**  ----------- 1

printf(“You entered 5”) **;** ----------- 2

- Statement 1 and 2 will act as **two different statements** , since null operator ( ; ) after if() statement gets the printf() operator out of the if block

- Hence , printf() is bound to get executed no matter if() gives true or false . And there **won’t be any error**

xcxc

- format of if-else :-

if(**condition** is true)

statement 1;

else

statement 2 ;

- We use **relational operators(< , > , <= , >= , == , !=)** to express the conditions in if-else statement

- ‘=’ is used for assignment whereas ‘==’ is used to compare two quantities

- We can use if() without else , but else must always be associated with an if()

- Any non-zero number is true , 0 is false .

- **‘;’** is a null statement . It doesn’t do anything on execution

- **Logical operators , &&(AND) , ||(OR) and !(NOT)** allow to get two or more

conditions

if(i=5)

statement ;

- **this statement won’t throw error**

- In an **else if** statement , the last else is optional

- **!** , this logical NOT operator reverses the result

- **!(NOT)** comes at the top in priority list of the operators

- **!** is an **unary** operator , since it needs only single operand and **rest are binary** operators

***if(!flag)*** is same as ***if(flag==0)***

***!(a <= b)*** *is same as* ***(a>b)***

***!(a>=b)*** *is same as* ***(a<b)***

**Conditional Operators**

- **?** and **:** constitute conditional operators .

- These are called as ***ternary operators*** as they take 3 arguments .

format :- **expression 1 ? expression 2 : expression 3**

**- If express 1 is true (non-zero) , then express 2 will be returned otherwise express 3 will be returned .**

**-** The limitation of conditional operators is that **only one C statement** can occur after **?** or **:**

***a>b ? g = a : g = b ;* 🡨 error , “Lvalue Required” ; to fix : a>b ? g = a : (g = b) ;**

- **sizeof()** is an operator which gives **number of bytes** occupied by an entity

**Loop Control**

- Loop control instruction is used to repeat a set of statements either a specified number of times or till a particular condition keeps satisfying .

- the variable used for loop control is called **‘loop counter ‘** or **‘index variable’**

- **for , while and do-while** are the kinds of loop control statements.

- **j = ++i** , first increments i then assigns the incremented value to j

- **while(++i<10) ,** first increments i then checks the condition

- **while(i++<10) ,** first checks the condition then increments

- **+= , -= , \*= , /= , %=** are called compound assignment operators .

- in **for loop** , **counter initialization , testing and incrementation** is done in a single line only , unlike in while loop .

- multiple initialization ,incrementation and testing could be done in for loop

example:- **for(i=1 , j = 2 ; i<=10 && j<=24 ; i++ , j+=3)**

- **break** statement is used to **terminate the loop** inside which it is .

- **continue** keyword **skips/abandons** rest/following instructions in the loop and goes for the next iteration of the loop.

- when **continue** statement is encountered the **control** is passed to the **beginning of the loop**

while with **break** , control passes to the **first statement after the loop** .

- Since **do-while** loop checks the condition after the execution of the statements , it executes its statements at least once even if the condition fails for the first time .

- **fflush()** is used to remove or flush out any data remaining in the **buffer** .

- we have to pass **stdin** as argument into the **fflush()** , since we have to flush out the standard input related buffer i.e **fflush(stdin)** .

- After taking the **number input** , the **scanf()** assigns the number to variable and **keeps the enter key unread in the Keyboard buffer** . Hence next time when we supply any character scanf () will read **the enter** from the buffer instead of taking character input . Hence to avoid this we use **fflush(stdin) .**

**-** sometimes , there might be a situation when in a loop the **number of iterations would be unknown** example: **while(x == 5) , while(y == ‘y’) ,** such loops are called **odd loops .**

- **usual uses :-**

while - to repeat something an **unknown number of times**

for - to repeat something a **fixed number of times**

do-while - to repeat something **at least once**

- **for(; ;)** and **while(1)** are infinite loops and **while()** is an error

***- what can we done using one loop can always be done using the other two***

**Case Control**

- **switch-case-default** control statement allows us to make a decision from the number of choices

- The switch statement is very useful for menu driven programs

- In the **switch()** , **float**,double expression can’t be tested (**only constant char , long int and int is used**)

- Switch-case works faster than if-else ladder

- break takes the control out of the switch() but continue doesn’t take the control to the beginning of switch .

- cases in the switch must always be unique

- **goto** keyword can take the control form any place to any other place within the function

- **exit()** from **<stdlib.h>** is a standard library function used to terminate the program execution

**Functions**

- Function is a self-contained block of statement that performs some tasks and returns back the evaluated result to the calling function.

- Basically , we create functions to avoid repeatedly rewriting the same codes and for the ease of debugging and designing

- Creating a function consists three tasks done , **Prototype declaration** , **Function Call** , **Function Definition** .

- Arguments passed to a function by the calling function are **Actual arguments** and that received by the function are **Formal arguments .**

- In C , program always begins with **main()**

- **Library functions** - printf() , scanf() etc , commonly required functions grouped together and stored in library files

- **User-defined** functions are freshly defined by the users.

- **return** statement transfers the control back to the calling function and terminates the current active function.

- A function can return only one value .

int a = 1 ;

printf(“%d%d%d”,a,++a,a++); 1 3 3(order of passing 1 to printf())

Output : 3 3 1 (order of printing)

**In C , during a function call , the arguments are passed from right to left .**

- Order of passing arguments in function is compiler-dependent not language-dependent

- Headerfiles contain the library-functions prototypes

- For example , <stdio.h> header file contains the prototypes of all the input/output functions , and prototypes of all mathematical functions are provided into the header file ‘math.h’

printf(“%d%d%d”,a,b);

printf(“%d”,a,b);

**Both the statements would be compiled without any error as printf() accepts variable arguments**

Pointers