A beginners guide to C language

-Saikat Mohanta

C is one of the most popular programming languages, originally developed in *1970s* by *Dennis Ritchie* at *Bell Telephone Laboratories* of USA. The language is called **C** because it was the successor to a language named **B**.

Some of the qualities of the programs written in C has the following qualities :-

1. *The programs written in C is **concise, efficient & fast**.*
2. *A large number of **standard functions** are available with*
3. *It can be used **both as high level and low level languages**.*
4. *The program is **machine independent**.*

In order to learn C language at first one must be acquainted with the following terms,

1.character set:

A-Z, **a-z**, **0-9**, **#**, **@**, **&**, **%**, **()**, *****, **;** , etc , some special combination of special characters such as **\b**,**\n**,**\t** (called escape sequence) etc.

*C **characters** are also be put together to form **C word** which are used in C programs as **key elements***. A C word can be classified as a ***keyword*** or as an ***identifier***.

A.Keyword:

There are some ***reserved words***, called ***keywords***, that has *standard predefined meaning* in C.

Meaning of these *keywords* cannot be *changed*.

Some of the *keywords* are :
auto, char, const, if, else,for, float, int, goto, return, void, while, case etc.

All keywords are written in *lowercase*. C is very ***case sensitive***. In C everything is written in lowercase letters, but uppercase can be used for symbolic names.

B.Identifiers:

*It refers to the name of various **program elements** such as **variables**, **names representing constants**, **functions** and **arrays**.*

These are *user defined names* and *consist of letters and digits* with the first character being a letter or an underscore. It cannot contain a **blank space** or **coma**.

Thus **sum**, **name**, **basic_pay**, **area, x0** all are valid *identifiers*. On the other hand
3x, **order-no**, **first boy** are not valid *identifiers*.

2. ***Constants:***

*C constants refer to **fixed values** which do not change during **execution** of a program.*

There are four basic type of C constants,

A. Integer constants: -

*It refers to an **integer-valued number**.*

Decimal point, blank space, coma, etc are not permitted between digits. *If necessary + and - signs can be used*. Examples of valid ***integer constants*** are: **122, +455,-273**
On the other hand **1,200 , 11.23, 5_4** etc are illegal *integer numbers*.

B. Floating point constants: -

It refers to ***numbers positive or negative*** *with* ***a decimal point*** *or* ***an exponent*** *or* ***both****.*

But it must not contain *blank space, coma,* etc. Illegal characters.
Examples of valid *floating point* numbers are : **1.5, -3.141, 2e-4, -1.2e+4.**
On the other hand **14, 1,245 , 3E15** are invalid floating point constants.
A floating point number is stored in 32 bits *with 6 digits of accuracy*. When more precision is required type **double** or **long double** can be used.

C. Character constant: -

A character constants contains ***a single character*** *enclosed within* ***a pair of single quotation marks***.

Examples are,
'A' , ';' , ' ' etc.
The last example is a *blank character*. The character constants have *integer values* known as ***ASCII values***.

D.String constant: -

A string constant is ***a sequence of characters*** *enclosed within* ***double quotation marks****.*

Here character may be *letters, numbers, special characters and blank space.* Examples are:
"student" , "try again" , "N" etc.

E.Symbolic constants:

A symbolic constant ***allows a name*** *to appear* ***in place of a numeric constants, a character constant or a string****.*

In some C programs a constant may appear at many places and may subjected to a frequent change.

In such cases it becomes convenient to use meaningful name for the constant using **#define directive**.

For example , **#define PI 3.1415927**

Give a name ***PI*** to the constant ***3.1415927***. when a program is compiled each occurrence of the name PI is replaced by the value **3.1415927**. Use of **#define** makes it easier to modify the program.

Note that there is ***no semicolon*** follows define because it is a ***pre processor compiler directive*** and ***not a statement***.

*Usually a **symbolic name** is written in uppercase letters to distinguish it from variables.*

#define *can also be used **to define a name** in place of **an expression or statement.*** For example,

#define CUBE(x) x*x*x

Gives a name **CUBE(x)** to the expression **x*x*x**.
If the program contains a statement

y=CUBE(ab);

The processor would expand this statement to

y=(ab*ab*ab);

3. ***Variables and arrays:***

*A variable is **an identifier** that may be **used to store value**.*

Unlike *constants variable* may take up different values during the *course of program execution*. However, the *data type* associated with the *variable can not change*. The *data type* related to a variable name must be declared at the beginning of the program.

*An **array** is a collective name of variables of the same data type.*

*The **individual variable** making up the array is called an **element** of the array.*

Array elements are denoted by the *array name followed by an integer index* enclosed within *rectangle brackets*. The *integer* or subscript always starts from ***0*** in C.

For example, in a ten element will be represented by roll[0], roll[1], ..., roll[9]. It is an example of one dimensional array. Arrays can be multidimensional. Array elements can be used in programs just like any other C variables.

Type declaration:

The data type of a variable used in a C program must be properly declared before it appears in executable statement. The syntax is,

<data type> <list of variables>;

For example, valid declarations are:

int a,b;
float c,d;
char c1,c2;

The first one declares **variables a,b** as ***integers***, the second one declares ***c,d as floating point type*** and the third one declares ***c1, c2 to be of character type***. Note every *statement* in C should end with a *semicolon(;) mark*.

Like other *variables* must be declared before they are used. For example,

int a[10];

Creates a one dimensional array named a having 10 elements named **a[0], a[1],⋯,a[9].**

Similarly,

float b[2][2];

Creates a two dimensional array named b having 2x2 elements named, **b[0][0], b[0][1], ⋯, b[1][1].**

Very often a variable is given an initial value at the time of type declaration. For example,

int b = 5;

Declares a integer type variable whose initial value is 5.

The statement,

int a[3] = (1, 2, 3);

Initializes **a[0] = 1, a[1] = 2, a[2] = 3.**

Similarly,

int b[2][2] = { {1, 2}, {3, 4} };

Initializes **b[0][0] = 1 , b[0][1] = 2 , b[1][0] = 3, b[1][1] = 4**.

A character type array can also be initialized. For example,

char text[8] = "student";

Initializes **text[0]="s", text[1]="t", text[2]="u", ⋯ , text[6]="t", text[7] ="\0".**

The *eighth element* would automatically put as *null character*(**\0**) which is automatically added at the end of the *string*.

4. ***Operators and expressions:***

*An operator is **a symbol** that tells the computer to perform certain **mathematical or logical manipulations of data** and **variables**.*

C language has a rich *set of operators*. *Operators usually form a part of mathematical or logical expressions*.

A. ARITHMATIC OPERATORS :

*+ addition, - subtraction, * multiplication, / division, % modulo division giving remainder of an integer division.*

B. RELATIONAL OPERATOR :

a>b a greater than b
a>=b a greater than equal to b
a<b a lesser than b
a<=b a lesser than equal to b
a==b a equal to b
a=b check whether a is equal to b
a!=b a not equal to b

C. LOGICAL OPERATORS:

&& logical AND operation
|| logical OR operation
! logical NOT operation

D.ASSIGNMENT OPERATORS:

They are used to assign the result of an expression to an identifier. The most common assignment operator in C is '=' . C also has a set of shorthand assignment operators that make the program easier to read, understand and more efficient.

Statement with simple Assignment operators	Equivalent statement with Shorthand operators
x=x+3;	x+=3;
x=x-5;	x-=5;
x=x*(n+2);	x*=(n+2);
x=x%3;	x%=3;

E.UNARY OPERATORS:

In C there are operators that act on a single operand.
These are called unary operators. The increment operator ++ adds 1 to the operand and the decrements operator -- subtracts 1 form the operand. The unary (-) minus operator negates the value of the operand. Thus ,

++x or x++ increments x by 1,
--x or x-- decrements x by 1,
-x negates the value of x.

Note that the increments or decrements operators may be used before or after the operand. They mean the same thing when used independently. But when used in expressions they may carry different meaning.
For example,
y=++x; means x is incremented by 1 and then the result is assigned to y.
But y=x++; means the value of x is first assigned to y and then x is incremented by 1.

PRIORITY OF OPERATIONS: (OPERATOR PRECEDIENCE)

While evaluating an expression involving more than one operators one may know the level of priority of different operations. The operator discussed so far are aranged below with decreasing order of priority from left to right:

1. Unary operators , 2. (*,/,%,+,-) , 3. relational operators, 4. equality operators, 5. &&, 6. || , 7. assignment operators.

5. Library functions:

A function in c language is a self-contained program segment that carries out some specified well defined task.

Every C program consists of at least one or more functions. C has a large number of *built in functions*, called *Library functions* which can be invoked in a program to perform various commonly used *operations or calculations.*

For example,

Function	log(x)	sqrt(x)	abs(x)	exp(x)	pow(x,y)	sin(x)	fmod(x,y)
Meaning	ln(x)	$\sqrt{x}$	$\|x\|$	e^x	x^y	sin x	Remainder of x/y

Library functions which are ***functionally similar*** are grouped together and stored in a *different library files* known as ***header files.***

In order to invoke a library function the concerned file name must be included at the beginning of a program. This is done by,

#include <file name.h>

For example,

#include <stdio.h> //for standard I/O library
#include <math.h> //for standard math library

A.Function main( ) :

Every C program must have the function main ( ) which may call(access) other functions.

Thus main ( ) is a *calling function*. Program *execution* begins with it. Other functions are *subordinate to this functions*.

C. Input and output functions:

Most programs take some data as input and display the processed data as output.

The commonly used *library functions* for *input and output* are,

getchar( ), putchar( ), scanf( ), printf( ), gets( ), puts( ) ,etc.

To use this *standard functions* it is necessary to include the *standard input-output header file* at the beginning of the program. The necessary statement is:

#include <stdio.h>

I) **Getchar( )** :-

It is a input function which can be used to read a character from a standard input device such as keyboard.
The syntax is,

Variable name = getchar( );

Where *variable name* refers to some previously declared *character variable*. When this statement is encountered the computer waits until a *single character input* is given with the help of keyboard. Then it assigns the *character value* to the *variable*.

II) **Putchar( )** :-

It is a single character output function.
It can be used to display a single character. For example,

putchar(ab);

Causes the current value *character variable* ab to be displayed on the *monitor*. The getch( ) & putchar( ) functions can be used repeatedly to read or output *multi character string*.

III) **Scanf( )** :-

Input data can be entered into the computer by using scanf( ) function.

It can be used to *enter* any combination of *numerical values*, *single character* and *strings* and *store* them as the values of appropriately listed variables.the general format is :

Scanf("control string", argument1, argument2,⋯);

The control string contains the format of data being input by the user.

Control string	meaning
%c	Data is a single character
%d	Data is a decimal integer
e, % f, % g	Data is a floating point item. e is used for representation of data in the exponent form.
%h	Data is short integer
%i	Either decimal, hexadecimal, octal integer
%o	Data is octal integer
%s	Data is a string
%u	Unsigned integer
%x	Hexadecimal integer
[ ]	String which includes blank space, special characters etc.

An example:

```
int x;
float y;
char student[20];
Scanf( "%d %f %s", &x, &y, student);
```

The symbol **&** indicates that the *argument* following it is *a pointer* that specifies the *address* or *location* in the *memory* where the *value received* from the keyboard is to be stored.

Note that an array name is not preceded by & since this is not a pointer to a memory location.

When the scanf( ) function is encountered the computer stops *execution* and waits for the *values of variables* to be typed in.

IV) **Printf( )** :-

It is used to get(print) output data from the computer onto a standard output device such as display monitor.

The general format for printf( ) is:

printf("control string" , arg1, arg2,⋯);

Here *control string* can contain the following information:

1. C*haracters* to be printed as such on the screen.
2. *Format* specifications for variables
3. *Escape sequence* characters
4. The arguments are the variables whose values are to be printed. For example,

printf("%d %f" , x, y);

It would output the current values of the integer variable x and floating point variable y.

Certain features can be used to control the alignment and spacing of printouts. Format specifications for integers is

%ωd where ω is an integer specifying the number of digits for the output.

%ω.p f where ω is the number of digits shown before and p is the number of digits shown after the decimal point.

For example,

```
float x = 15.32145;
printf( "%3.3f" ,x );
```

Will print **015.321**

Immediately after the % sign we can include flag which affects the appearance of the output. For example, a - sign left justifies the data, a 0 after % sign causes the leading zeroes to appear instead of leading blanks.

printf() function can be used to display a message only. For example,

printf("! hello world !");

Will print **!hello world !**

Escape sequences can be used with printf( ) function to perform special tasks.

For example, **printf("hello\nworld");**

Will print **hello**
World

Escape sequence	effect
\a	Produce bell sound backspace
\b	backspace
\n	Start new line
\r	Insert carriage return
\'	Print apostrophe
\"	Print double quotes
\?	Print question mark
\\	Print back slash

V) **Getch()** :-

The function getchar() reads a single character.

It *returns* the *character* that you have *typed* but *requires* the ***enter key*** to be pressed following the character that you have typed.

In some cases we want a *function* that will read a *single character* the instant it is typed without waiting for the *enter key* to be pressed. The *function getch( )* may serve the purpose.

5. <u>User defined functions:-</u>

apart from pre defined functions available in C we can define functions according to our requirements.

These are used when a *set of instructions* are used repeatedly.
The general format of a function is:

```
function name    function type ( list of parameters )
{
    executable statements;
    ……….
    return statement;
}
```

An example,

```
int add ( int x, int y)
{
    int s;
    s = x + y;
    return (s);
}
```

When the *compiler* encounter the *function* **add** the control is transferred to the function. After execution of the function a value is returned to the calling function. If there is no value to be returned, then the return statement is not necessary.

A function which does not return any value is defined as,
Void function **name** (**arguments**)

Sometimes instead of the *function* **main( )** we use **void main( )** which means that the *function* does *not return* any value to the *operating system*.////

6. Logical decision and branching:-

When a *program* is given to a *computer* it starts *execution* from the *first statement*. After the *first statement* control goes to the *second statement* and so on. However we can change this *normal flow of control* based on certain conditions. Here we shall discuss some of such *statements*.

A. Simple if( ) statement :

It is the ***simplest control statement*** *or* ***control structure.***

Its general form is,

```
if( test expression )
{
    Statement1;
    Statement3;
    ……….
}
Statement2;
```

Where the *test expression* is a *logical condition* and the statement is *executable statement*. If the *test expression* is true the statement1 will be executed and then goes to the next statement3. Otherwise the statement 1,3 will be skipped and control directly goes to the statement2.
For example,

```
int angle;    //declaration of variable and type
……….
scanf( “%d” ,&angle);    //input of the angle data
```

```
if(angle = 90)  //checks whether it is 90
{
    printf( "right angle" ); //for data output
}

Printf( "%d" , angle );//for data output
```

The above program segment inputs an angle value, checks whether it is 90 degree, if it is 90 degree, it prints right angle. Otherwise it prints the input angle.

B.if-else statement():

If-else *statement is an* ***extension*** *to the simple* ***if*** *statement.*

It is used to carry out a *logical test* and then take one of the two *possible actions* depending on the outcome of the test.
The general form is,

```
if( test expression )
{
    Statement1;
}

else
{
    Statement2;
}
```

The statement1 is executed when test expression is true and statement2 is executed when test expression is false.
For example,

```
float marks;
……
scanf( "%f" , &marks);
```

```
if(marks>35)
{
    printf( “pass” );

}

else
{
    printf( “fail” );

}
```

If the marks is greater than equal to 35, then prints pass. Else prints fail.

C.Nested if-else :-

When a series of decisions are involved one can use more than one if-else statements in nested form, ie. One inside another.

Example :

```
if( a>b )
{
    if( c>d )
    {
        x = p;
    }

    else
    {
        x = q;
    }
}
```

```
else
{
    x = r;
}
```

D.The goto statement:-

It is used to ***alter the normal sequence*** *of* ***program control*** *from one point to another in the program.*
It requires a *label* to i*dentify the place* where the *control is to be transferred*. The general form of *goto label* is:

```
goto label
……….
label : statement
```

Here a *label* may be any *valid variable* name with a *colon*.
For example,

```
#include <stdio.h>

main( )
{
    START : printf( "hello world" );
    goto START;
}
```

Will show,

```
hello world
hello world
hello world
…….
```

7.Looping :-

Sometimes we require to repeat some portion of the program either a specified number of times or until some particular condition is satisfied.

Such repetitive operation is done through looping. In C language *looping* can be done by using ***while***, ***do-while*** and ***for*** statements.

A.While loop:

The general format of the while loop is,

```
While( test condition )
{
    Body of the loop
    ……….
}
```

Here the *body of the loop* is *executed repeatedly* as long as the *condition is true*. When the *test condition is false* the program *control jumps out* of the loop to the next part of the program.
Example:

```
…….
int sum = 0;
int n=1;
while( n<= 100)
{
    sum+ = n;
    n+ = 1;
    …….
}
```

Note the above program segment evaluates the sum of
1 + 2 + 3 + ⋯ +100.

B.do-while loop:

This is similar to while loop except that the test condition is checked at the end of the loop.
So there is guarantee that the do while loop will be executed
At least once. The general structure is,

```
do
{
   Body of the loop
}
while ( test condition )
```

Example:

```
int num = 0;
do
{
    printf( "%d\n" , num);
    ++num;
}
While ( num<=20 )
```

Outputs,

```
hello world
hello world
hello world
……….
```

Example:

```
do
{
    int i = 0;
    sum = sum + i;
    i++;
}
while ( i != 10 )
```

The above program segment prints 0 to 10 each number in new line.

C.for loop:

It is the most common looping statement.
It general form is,

for(initialization;testcondition;increments or decrements)
{
Body of the loop
}

For example,

……
int sum = 0;
for(int i = 1; i<=20 ; i++)
{
sum = sum + i;
……
}

Note that the above program segment calculates the the value of **1 + 2 + 3 + ……+20.**

D.continue and break statement :

The continue statement is used in a loop to skip the remainder of the current cycle through a loop and continue the next cycle of the loop.

The general format is ,

continue;

The break statement is used in a loop to terminate the loop and transfer the control out of the loop.
The break statement is written simply as,

break;

8.Comments:-

Comments (remarks) may appear anywhere within a C program. They are placed within the *delimiters* **/* and */** .
They are helpful in understanding the program. The *compiler* has nothing to do with such *comments*. The *comments* which are placed inside the *delimiters* simply does to *compile* at all.

Saikat Mohanta
Bongaon
West Bengal