Understanding C Basics-I *

K.M.Shafi†

December 9, 2016

1 Introduction

C is the mother of computer languages by its influence still lasting and has also became the *lingua-franca* among almost all the computer scientists and professionals. Developed in 1972 by Dennis Ritchie at Bell Labs (AT&T), it got its first ANSI [1] standardization in 1989 commonly referred as C89.Later, it was amended and then re-standardized in 1999 and is refereed as C99 containing all the features of C89 plus standardized amendments .Nowadays, programming in C is essential part of the computer curriculum at secondary and bachelors level.

2 Data Types

What is data type? It refers to what type/nature of data a particular computer language is able to represent and process. As every thing at the hardware level on the computer is stored as 0s and 1s,it is the language itself and programmer which give sense to those 0s and 1s. Whenever you need to do some processing and representation with particular data, you first need to sort out what type of that data is. It can be a name of an item, or a value representing some quantity and that value can be a positive, integer or fractional number or it can be photograph or a signature or some video footage. At the hardware level, all these types of data are stored as 0s and 1s but at the programmers level every data type needs different treatment and

*This tutorial is originally intended for the PGDCA students batch 2015-16 of DDE, KU

†mshafi710@gmail.com

[1] American National Standards Institute

has well defined properties and operations associated with it like we can't add two addresses but we can add two salaries as the former operation does not make sense.
If the language inherently supports some data types then those are categorized as primitive or basic or primary data types. Basic data types doesn't always suffice for representation and processing needs of program. In that case, the programmer tries to create or define its own data types, generally known as user defined data types.
The basic data types supported by the C language are integers, floating point and characters.

2.1 Integers

Integer data types and its variants are used to handle numeric data which is non fractional. To instruct a computer to have a space of integer type, basic C statement would like *int x;* where *int* is the keyword and x is the name (variable) of that space in the memory reserved for your program and can be utilized for storing and processing information. Keyword means it is reserved and has a predefined meaning. x can be replaced by any other valid name as per the rules of C Language (see Variables subsection below). Every basic statement of C language is terminated by a semicolon. Range of values that can be stored in the variable (here x) is determined by its data type which is *int* here. On 32-bit compilers [2] , the variable declared as *int* takes 32 bits and therefore by the formula of -2^{31} to $+2^{31}-1$ it can hold any value between -2147483648 to + 2147483647.
Statements like *x=0; x=710; x=-999999; x=987654321; x=-2147483648* are all valid assignments to x; where as *x=2147483648; x= -30000000000* are invalid as they are out of the range. Nature of *int* data type can be delimited or extended by using some modifiers using *short*, *long*, and *unsigned* keywords. Statement like *unsigned int x;* gets same 32 bit memory but this time it can be used only for storing non-negative numbers with range doubled i.e 0 to $2^{32}-1$, while as statement like *short int x;* will take only 16 bits with its range shortened from -2^{15} to $+2^{15}-1$.

2.2 Floating Point

In C, real numbers whether in fractional or exponential form can be handled either through *float* or *double* data type . They differ in the number of bytes allocated to them and the range. *double* can be extended by *long* modifier to accommodate real numbers with greater precision. 710.5 , -710.5 ,

[2] 32-bit compiler means

710.5 are examples of fractional while as + 7.0e+5, .1E+3 are of exponential form.

2.3 Characters

Character data type is used to represent any character like a letter of an English alphabet, digit, or a special symbol like $, %, # etc enclosed in single quotes. Variable declared as character (by default *signed*) gets only one byte allocated, hence restricting its range from -128 to +127. An *unsigned* character can have value from 0 to 255 therefore can represent only ASCII [3] and Extended ASCII characters

2.4 Data Type Modifiers

Data types can be modified in C to suit a particular need. Although, all numeric data type are signed by default, we can explicitly mention that using modifier *signed* as prefix at the time of declaration. To make variable unsigned, it is mandatory to use *signed* keyword before data type (only for *int*, *char*, *long*). Various modifiers used are *signed*, *unsigned*, *long*, *short*. *long* modifier is used only with *int* and *double* data types to increase their range, while as *short* is used for only integers to decrease their ranges. The table below shows the size and range of the frequent used data types and their modifiers.

3 Constants & Variables

3.1 Constants

A constant is a literal in C representing some value which cannot be changed. For instance, 710 is an integer constant, 8.2E7 is a floating point constant, '$', 'A','%' is a character constant. They can be assigned to a variable or can be used directly in an expression. Character constants are always enclosed in single quotes in C.

3.2 Variables

Technically, variable refers to name given to a memory location. It can be a one byte location or multi-byte depends upon the data type of which the

[3] ASCII means Ameraican Standard Code for Information Interchange

Type	Size	Range
char	8	-128 to +127
unsigned char	8	0- 255
signed char	8	same as char
int	32	-2^{31} to $+2^{31}-1$
unsigned int	32	0 to $+2^{32}-1$
signed int	32	same as int
short int	16	-32768 to +32767
unsigned short int	16	0 to 65335
signed short int	16	same as short int
long int	32	same as int
signed long int	32	same as signed int
unsigned long int	32	same as insigned int
float	32	3.4e -38 to 3.4e +38
double	64	1.7e-308 to 1.7e+308
long double	80	3.4e-4932 to 1.1 e +4932

Table 1: Size and range of data types in C on a 32 bit machine

variable was declared. With a particular variable, many things are associated with it either implicitly or explicitly. A variable has name, value, address, data type, storage class. Address means where in the memory a variable has got its space and storage class specifies its default value, its scope i.e., its visibility within the program, its lifetime i.e., to what extent it can exist in the memory during the execution of program. Program can change the variable's value but not the address or storage class. In C, every variable used needs to be declared at the start of the program.

3.2.1 Variable Naming Rules

Variable name shall always start with a letter or underscore and may be followed by letters or digits or underscore. Variable name cannot contain spaces or special symbols. More important it can not be a keyword. Some compilers restrict the length of variable name up to 255 characters.

1. *rollno, _rollno, rollno1, rollNo, roll_no1* are examples of valid variable names.

2. *6rollno, roll no,%rollno* are examples of invalid variable names due to reasons mentioned above.

Less than	<
Less or equal to	<=
Greater than	>
Greater or equal to	>=
Equal to	==
Not equal to	!=

Table 2: Relational Operators

4 Operators

Operators represent or refer to operations which can be performed on data. The data (constants,variables, expressions) on which they act are called operands. Operators need to be in consonance with their operands. Depending upon the number of operands they need to perform upon, operators can be categorized as unary, binary and ternary. Binary Operators needs two operands to perform, ternary operator need three and unary need one. Well defined combination of operators and operands *i.e., as per the rules of the C language* make an expression which on evaluation give a result which can be used for storing, displaying or used in evaluating other expression. C provides large range of basic operators which are discussed below.

4.1 Arithmetic Operators

$+, -, *, /, \%$ for addition, subtraction,multiplication,division and modulus operations respectively. These are valid for only those data types which represent numerical values. For instance, simple statement for performing multiplication on two integers and storing the result in third can be like this z=x*y; in which the values of x and y are multiplied and the result is put into z without changing the values of x and y. Similarly *z=x%y ;* is the C statement in which y divides x and puts the remainder in z instead of quotient. For extracting the quotient, operator / is used (only in case of integer operands).% (modulus) operator can't be used with non-integer data types.

4.2 Relational Operators

Relational operators perform arithmetic comparison between the values of its operands and give the result as either 0 or 1.The relational operators supported by the C language are given in the following table.

Logical AND	&&
Logical OR	||
Logical NOT	!

Table 3: C logical operators

If the variable x holds 20 and y 30, then $x < y$ will give 1 as result and $y > 100$ will give 0. In other words, the first expression says 'is x less than y', which it is and hence 1 representing truth value of true. Similarly 2nd expression says 'is y greater than hundred', which it is not hence 0 representing truth value of false.Relational operators can be used with any numerical data type.

4.3 Logical Operators

Normally, logical operators in C are used to combine the multiple relational expressions. C provides three operators for such case as shown in table below.

The && and || are binary operators i.e., involving two operands or expressions while as ! is unary. The result of logical expression involving && will yield true value only in case if all of its operands are true other wise in all other cases it will yield false value only. Similarly, logical expression of || will yield false if all of its operands are false other wise in all other cases it will yield true. For instance the logical expression $(5 < 6)\&\&(100 < 5)$ will yield false because $100 < 5$ yields false value and trueness of $5 < 6$ does not make this expression true. Similarly, $(5 < 6)||(100 < 5)$ yields true because at least one of the subexpression is true. In case of ! operator $!((5 < 6)||(100 < 5))$ expression will yield false while as $(5 < 6)\&\&(100 < 5)$ will yield true because negation operator (!) negates the truth value of its operand on which it acts.

4.4 Assignment Operators

C provides number of assignment operators like =, +=, -=, *=, /=, %=. Assignment operators are binary in nature taking the value of expression to its right side and puts its value to the variable on left side. For instance $a = 5*20+30$; expression will assign the value of 130 to variable a. The above given operators except = are also called compound assignment operators as they are a little bit different from the normal assignment operator. For instance, $a+ = 10$; will add the value of 10 to the a and put the resulted value back into a. Similarly, $a* = 10$; will multiply the value of a by 10

and put the resultant value back into a. Same is the case with all the given operators.

4.5 Unary Operators

Unary operators take only one operand to act upon. C provides different unary operators like -(unary minus) which multiplies it contents by -1. C provides two different special operators viz., increment ++ and decrement −−. ++ increases the value of its operand by one and −− decreases by 1. Both ++ and −− has two versions viz., pre ++, post ++, pre −−, post −−. Both these two versions will act differently when used along with some expression otherwise if used as standalone there is no difference between the two versions. Table below tries to explain all the version of these operators. Assume variable a and b has already been initialized with values 10 and 20 respectively. //

4.6 Precedence & Associativity

If an expression contains multiple operators, then the overall result may be different depends upon the order of evaluation of individual operators in the expression. In order to have accurate value all the time,order of evaluation is standardized and thus precedence comes into play. Precedence tells about which operator to evaluate first. But another situation arises when in an expression there are multiple operators having same precedence. To overcome such situation, associativity takes the role. It tells from which side to evaluate when there are multiple operators of same precedence.

5 Writing your First C program

5.1 Basic Structure of C Program

Execution of C program always starts from special function called *main()*. A function in C is self contained block of statements enclosed in curly braces and given a unique name. C program can have number of functions of different names but can have only one *main()* function. Concept of function will be discussed in upcoming tutorials.

```
int main()
{
<C language statements>
}
```

Expression	**Value**	**Explanation**
a++;	a gets 11	post ++ increments the value of a by 1.
++a;	a gets 11	pre ++ increments the value of a by 1.
$a--$;	a gets 9	post −− decrements the value of a by 1.
$--a$;	a gets 9	pre −− decrements the value of a by 1.
$b = a++$;	a gets 11 but b gets only 10	Here the value of a i.e., 10 is first assigned to b then value of a is post incremented by 1;
$b = ++a$;	a gets value 11 and b gets also 11	Here the value of a i.e., 10 is pre-incremented to 11 and then the same value of a is assigned to b.
$b = a--$;	a gets 9 but b gets 10	Here the value of a i.e., 10 is first assigned to b then value of a is post decremented by 1;
$b = --a$;	a gets value 9 and b gets also 9	Here the value of a i.e., 10 is pre decremented to 9 and then the same value of a is assigned to b.

Table 4: Examples of post/pre increment and decrement operators

Table 5: Precedence and associativity of some commonly used operators. For other opeartors check Appendix A

Description	**Operator**	**Associativity**
Unary Minus	-	Right to Left
Increment/Decrement	++ –	Right to Left
Negation	!	Right to Left
Multiplication	*	Left to Right
Division	/	Left to Right
Modulus	%	Left to Right
Addition	+	Left to Right
Subtraction	-	Left to Right
Less Than	<	Left to Right
Less than or equal to	<=	Left to Right
Greater than	>	Left to Right
Greater than or equal to	>=	Left to Right
Equal to	==	Left to Right
Not equal to	! =	Left to Right
Logical AND	&&	Left to Right
Logical OR	||	Left to Right
Assignment	= *= /=%= += -=	Right to Left

6 Basic I/O Statements

6.1 printf

Normally printf statement is used to put the messages and results of your program on the screen of computer. To use printf or scanf function in a program, one must use *#include < stdio.h >* statement (*compiler directive*) at the start of program. This directive tells the compiler (preprocessor) to include the stdio.h header file as part of the program at time of compilation as that file contains the details about functionality of printf function. The syntax of printf function is: Basic syntax:

```
printf("format string", variable1, variable2);
```

"format string" can contain direct text message or format specifier(s) referring to value(s) of variable(s) specified correspondingly after the message. Different format specifiers like %d for an integer in decimal format, %f for floating point type, %c for character type, %l for long type,%u for unsigned integer are used normally. E.g., *printf("programmers never get tired");* will display **programmers never get tired** on the screen while as *printf("%d%f",a,b) will display the values of variables a and b which are supposed to be of type int and* ***float***

6.2 scanf

scanf is the versatile statement used to get the input from user through keyboard. the basic syntax of the scanf statement is

```
scanf("format string", &variable1,variable2,...);
```

"format string" specifies what type of data the function is expecting from user and how much.The list of variables followed like &variable1, &variable2 specify the address in memory where the input received is to be kept. Program below demonstrates the use of print and scanf statement

```
#include<stdio.h>
int main()
{
int x;
char y;
float z;
printf("Enter tha values for x , y and z");
```

```
scanf("%d%c%f",&x,&y,&z);
printf("the value of x=%d y=%c and z=%f", x,y,z);
return 0;
}
```

Output of the program will be
Enter the values for x , y and z
9 @ 3.14
the value of x=9 y=@ and z=3.14

7 Comments

Comments are the statements which are embedded in the actual code to increase its readability and maintainability. Compiler skips those statements which are marked as comments. In C, comments are put using opening and closing pair of slash and asterisk as shown below.

```
int main(){
/*printf("Casual programmers");*/
printf("Die hard Programmers!");
return 0;
}
```

This code snippet will only display *Die hard Programmers!* and not *Casual programmers* as it is placed in comments and will be ignored by compiler.