LECTURE NOTE

on

PROGRAMMING IN “C”

COURSE CODE: MCA 101

By

Asst. Professor Mrs Etuari Oram
Asst. Prof. Mr Bighnaraj Naik

SYLLABUS

Module –I

C Language Fundamentals.
Character set, Identifiers, keyword, data types, Constants and variables, statements, expression, operators, precedence of operators, Input-output, Assignments, control structures decision making and branching.

Module -II

Arrays, Functions and Strings: Declaration, manipulation and String – handling functions, monolithic vs. Modular programs, user defined vs. standard functions, formal vs. actual arguments, function – category, function prototypes, parameter passing, recursion, and storage classes: auto, extern, global, static.

Module –III

Pointers, Structures, Unions, File handling:

Pointer variable and its importance, pointer arithmetic, passing parameters, Declaration of structures, pointer to pointer, pointer to structure, pointer to function, union, dynamic memory allocation, file managements.

*Under revision

CONTENTS

Lecture 13: String library functions

Lecture 14: functions, categories

Lecture 15: functions categories cont..

Lecture 16: Actual arguments and Formal arguments, call by value call by reference

Lecture 17:local, global, static variable

Lecture 18: monolithic vs modular programming, Storage classes

Lecture 19:storage class cont.., pointer

Lecture 20: pointer comparison, increment decrement

Lecture 21: precedence level of pointer, pointer comparison

Lecture 22: pointer to pointer, pointer to structure

Lecture 23: pointer initialization, accessing elements

Module: 3

Lecture 24: size of Structure in, array vs structure, array within structure

Lecture 25: passing structure to function, Nested Structure

Lecture 26: Union

Lecture 27: nesting of unions, dynamic memory allocation

Lecture 28: dynamic memory allocation conti…

Lecture 29: dynamic array, file

Lecture 30: file operation

Lecture 31: file operation on string

Lecture 32:

Lecture 33:

Lecture Note: 1

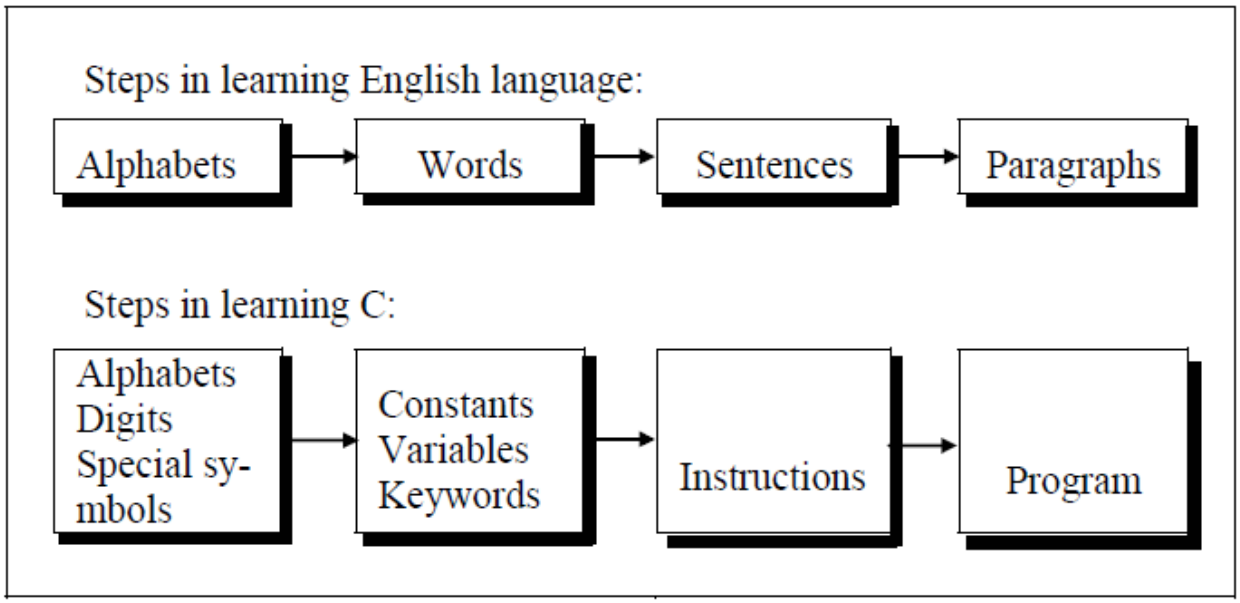
Introduction to **C**

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc

ANSI C standard emerged in the early 1980s, this book was split into two titles: The original was still called ***Programming in C***, and the title that covered ANSI C was called ***Programming in ANSI C***. This was done because it took several years for the compiler vendors to release their ANSI C compilers and for them to become ubiquitous. It was initially designed for programming UNIX operating system. Now the software tool as well as the C compiler is written in C. Major parts of popular operating systems like Windows, UNIX, Linux is still written in C. This is because even today when it comes to performance (speed of execution) nothing beats C. Moreover, if one is to extend the operating system to work with new devices one needs to write device driver programs. These programs are exclusively written in C. C seems so popular is because it is **reliable**, **simple** and **easy** to use. often heard today is – "C has been already superceded by languages like C++, C# and Java.

Program

There is a close analogy between learning English language and learning C language. The classical method of learning English is to first learn the alphabets used in the language, then learn to combine these alphabets to form words, which in turn are combined to form sentences and sentences are combined to form paragraphs. Learning C is similar and easier. Instead of straight-away learning how to write programs, we must first know what alphabets, numbers and special symbols are used in C, then how using them constants, variables and keywords are constructed, and finally how are these combined to form an **instruction**. A group of instructions would be combined later on to form a **program**. So



a computer ***program*** is just a collection of the instructions necessary to solve a specific problem. The basic operations of a computer system form what is known as the computer's ***instruction set.*** And the approach or method that is used to solve the problem is known as an ***algorithm***.

So for as programming language concern these are of two types.

1) Low level language

2) High level language

Low level language:

Low level languages are **machine level** and **assembly level language**. In machine level language computer only understand digital numbers i.e. in the form of 0 and 1. So, instruction given to the computer is in the form binary digit, which is difficult to implement instruction in binary code. This type of program is not portable, difficult to maintain and also error prone. The **assembly language** is on other hand modified version of machine level language. Where instructions are given in English like word as ADD, SUM, MOV etc. It is easy to write and understand but not understand by the machine. So the translator used here is assembler to translate into machine level. Although language is bit easier, programmer has to know low level details related to low level language. In the assembly level language the data are stored in the computer register, which varies for different computer. Hence it is not portable.

High level language:

These languages are machine independent, means it is portable. The language in this category is Pascal, Cobol, Fortran etc. High level languages are understood by the machine. So it need to translate by the translator into machine level. A translator is software which is used to translate high level language as well as low level language in to machine level language.

Three types of translator are there:

Compiler

Interpreter

Assembler

Compiler and interpreter are used to convert the high level language into machine level language. The program written in high level language is known as source program and the corresponding machine level language program is called as object program. Both compiler and interpreter perform the same task but there working is different. Compiler read the program at-a-time and searches the error and lists them. If the program is error free then it is converted into object program. When program size is large then compiler is preferred. Whereas interpreter read only one line of the source code and convert it to object code. If it check error, statement by statement and hence of take more time.

Integrated Development Environments (IDE)

The process of editing, compiling, running, and debugging programs is often managed by a single integrated application known as an Integrated Development Environment, or IDE for short. An IDE is a windows-based program that allows us to easily manage large software programs, edit files in windows, and compile, link, run, and debug programs.

On Mac OS X, CodeWarrior and Xcode are two IDEs that are used by many programmers. Under Windows, Microsoft Visual Studio is a good example of a popular IDE. Kylix is a popular IDE for developing applications under Linux. Most IDEs also support program development in several different programming languages in addition to C, such as C# and C++.

Lecture Note: 2

Structure of C Language program

1) Comment line

2) Preprocessor directive

3) Global variable declaration

4) main function( )

```
        {
          Local variables;
  Statements;
  }
  User defined function
 }
}
```

Comment line

It indicates the purpose of the program. It is represented as

/*……..*/

Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed within the decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & character constant.

Preprocessor Directive:

#include<stdio.h> tells the compiler to include information about the standard input/output library. It is also used in symbolic constant such as #define PI 3.14(value). The stdio.h (standard input output header file) contains definition &declaration of system defined function such as printf( ), scanf( ), pow( ) etc. Generally printf() function used to display and scanf() function used to read value

Global Declaration:

This is the section where variable are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function :

main()

It is the user defined function and every function has one main() function from where actually program is started and it is encloses within the pair of curly braces.

The main( ) function can be anywhere in the program but in general practice it is placed in the first position.

Syntax :

```
  main()
{
……..
……..
……..
}
```

The main( ) function return value when it declared by data type as

```
 int main( )
{
return 0
```

}

The main function does not return any value when void (means null/empty) as

void main(void) or void main()

{

printf (“C language”);

}

Output: C language

The program execution start with opening braces and end with closing brace.

And in between the two braces declaration part as well as executable part is mentioned. And at the end of each line, the semi-colon is given which indicates statement termination.

/*First c program with return statement*/

```
#include <stdio.h>

int main (void)

{

printf ("welcome to c Programming language.\n");

return 0;

}
```

Output: welcome to c programming language.

Steps for Compiling and executing the Programs

A compiler is a software program that analyzes a program developed in a particular computer language and then translates it into a form that is suitable for execution

on a particular computer system. Figure below shows the steps that are involved in entering, compiling, and executing a

computer program developed in the C programming language and the typical Unix commands that would be entered from the command line.

Step 1: The program that is to be compiled is first typed into a *file* on the computer system. There are various conventions that are used for naming files, typically be any name provided the last two characters are **".c"** or file with extension .c. So, the file name **prog1.c** might be a valid filename for a C program. A text editor is usually used to enter the C program into a file. For example, vi is a popular text editor used on Unix systems. The program that is entered into the file is known as the ***source program*** because it represents the original form of the program expressed in the C language.

Step 2: After the source program has been entered into a file, then proceed to have it compiled. The compilation process is initiated by typing a special command on the system. When this command is entered, the name of the file that contains the source program must also be specified. For example, under Unix, the command to initiate program compilation is called **cc**. If we are using the popular GNU C compiler, the command we use is **gcc**.

Typing the line

gcc prog1.c or cc prog1.c

In the first step of the compilation process, the compiler examines each program

statement contained in the source program and checks it to ensure that it conforms to the syntax and semantics of the language. If any mistakes are discovered by the compiler during this phase, they are reported to the user and the compilation process ends right there. The errors then have to be corrected in the source program (with the use of an editor), and the compilation process must be restarted. Typical errors reported during this phase of compilation might be due to an expression that has unbalanced parentheses (**syntactic error**), or due to the use of a variable that is not "defined" (**semantic error**).

Step 3: When all the syntactic and semantic errors have been removed from the program, the compiler then proceeds to take each statement of the program and translate it into a "lower" form that is equivalent to assembly language program needed to perform the identical task.

Step 4: After the program has been translated the next step in the compilation process is to translate the assembly language statements into actual machine instructions. The assembler takes each assembly language statement and converts it into a binary format known as ***object code***, which is then written into another file on the system. This file has the same name as the source file under Unix, with the last letter an "**o**" (**for *object***) instead of a "**c**".

Step 5: After the program has been translated into object code, it is ready to be ***linked.*** This process is once again performed automatically whenever the cc or gcc command is issued under Unix. The purpose of the linking phase is to get the program into a final form for execution on the computer.

If the program uses other programs that were previously processed by the compiler, then during this phase the programs are linked together. Programs that are used from the system's program ***library*** are also searched and linked together with the object program during this phase.

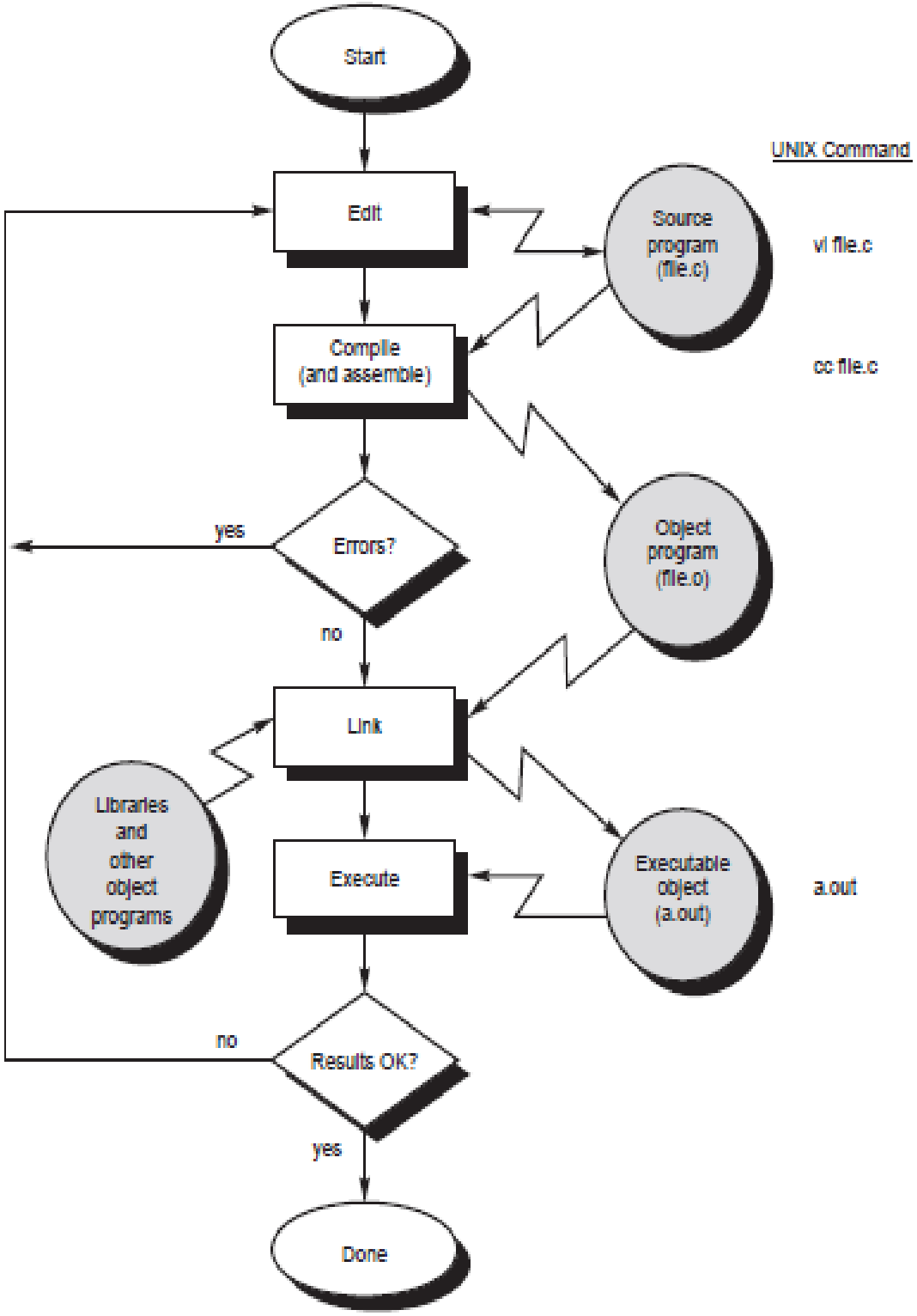
The process of compiling and linking a program is often called ***building*.**

The final linked file, which is in an executable ***object*** code format, is stored in another file on the system, ready to be run or ***executed*.** Under Unix, this file is called **a.out** by default. Under Windows, the executable file usually has the same name as the source file, with the c extension replaced by an exe extension.

Step 6: To subsequently execute the program, the command **a.out** has the effect of *loading* the program called **a.out** into the computer's memory and initiating its execution.

When the program is executed, each of the statements of the program is sequentially executed in turn. If the program requests any data from the user, known as ***input***, the program temporarily suspends its execution so that the input can be entered. Or, the program might simply wait for an ***event***, such as a mouse being clicked, to occur. Results that are displayed by the program, known as ***output*****,** appear in a window, sometimes called the ***console***. If the program does not produce the desired results, it is necessary to go back and reanalyze the program's logic. This is known as the ***debugging phase***, during which an attempt is made to remove all the known problems or ***bugs*** from the program. To do this, it will most

likely be necessary to make changes to original source program.



/* Simple program to add two numbers……*/

```
#include <stdio.h>
int main (void)
{
int  v1, v2, sum;                    //v1,v2,sum are variables and int is data type declared
v1 = 150;
v2 = 25;
sum = v1 + v2;
printf ("The sum of %i and %i is= %i\n", v1, v2, sum);
return 0;
}
```

Output:

The sum of 150 and 25 is=175

Lectu re Note: 3

Character set

A character denotes any alphabet, digit or special symbol used to represent information. Valid alphabets, numbers and special symbols allowed in C are

Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

The alphabets, numbers and special symbols when properly combined form constants, variables and keywords.

Identifiers

Identifiers are user defined word used to name of entities like variables, arrays, functions, structures etc. Rules for naming identifiers are:

1) name should only consists of alphabets (both upper and lower case), digits and underscore (_) sign.

2) first characters should be alphabet or underscore

3) name should not be a keyword

4) since C is a case sensitive, the upper case and lower case considered differently, for example code, Code, CODE etc. are different identifiers.

5) identifiers are generally given in some meaningful name such as value, net_salary, age, data etc. An identifier name may be long, some implementation recognizes only first eight characters, most recognize 31 characters. ANSI standard compiler recognize 31 characters. Some invalid identifiers are 5cb, int, res#, avg no etc.

Keyword

There are certain words reserved for doing specific task, these words are known as **reserved word** or **keywords.** These words are predefined and always written in lower case or small letter. These keywords cann't be used as a variable name as it assigned with fixed meaning. Some examples are **int, short, signed, unsigned, default, volatile, float, long, double, break, continue, typedef, static, do, for, union, return, while, do, extern, register, enum, case, goto, struct, char, auto, const etc.**

data types

Data types refer to an extensive system used for declaring variables or functions of different types before its use. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. The value of a variable can be changed any time.

C has the following 4 types of data types

basic built-in data types: int, float, double, char

Enumeration data type: enum

Derived data type: pointer, array, structure, union

Void data type: void

A variable declared to be of type int can be used to contain integral values only—that is, values that do not contain decimal places. A variable declared to be of type float can be used for storing floating- point numbers (values containing decimal places). The double type is the same as type float, only with roughly twice the precision. The char data type can be used to store a single character, such as the letter *a*, the digit character *6*, or a semicolon similarly A variable declared char can only store character type value.

There are two types of type qualifier in c

Size qualifier: short, long

Sign qualifier: signed, unsigned

When the qualifier unsigned is used the number is always positive, and when signed is used number may be positive or negative. If the sign qualifier is not mentioned, then by default sign qualifier is assumed. The range of values for signed data types is less than that of unsigned data type. Because in signed type, the left most bit is used to represent sign, while in unsigned type this bit is also used to represent the value. The size and range of the different data types on a 16 bit machine is given below:

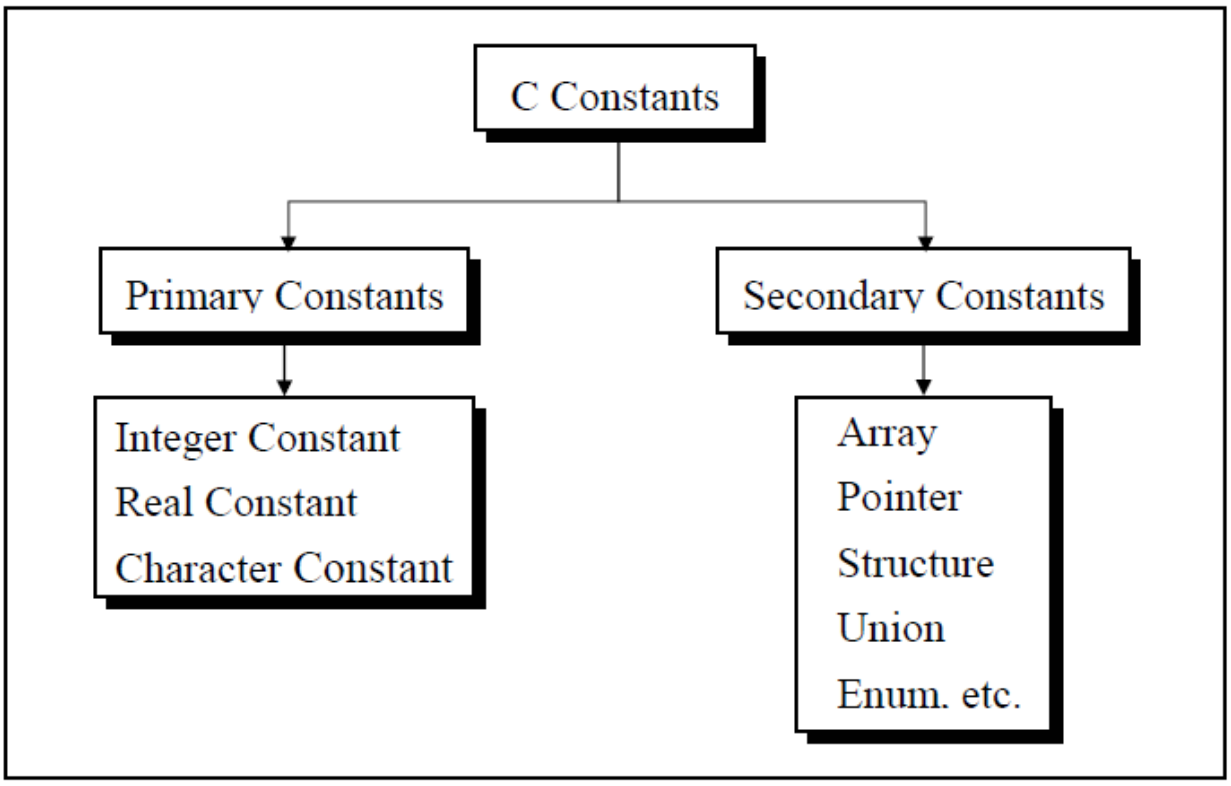
Basic data type	Data type with type qualifier	Size (byte)	Range
char	char or signed char Unsigned char	1 1	-128 to 127 0 to 255
int	int or signed int unsigned int short int or signed short int unsigned short int long int or signed long int unsigned long int	2 2 1 1 4 4	-32768 to 32767 0 to 65535 -128 to 127 0 to 255 -2147483648 to 2147483647 0 to 4294967295
float	float	4	-3.4E-38 to 3.4E+38
double	double Long double	8 10	1.7E-308 to 1.7E+308 3.4E-4932 to 1.1E+4932

Constants

Constant is a any value that cannot be changed during program execution. In C, any number, single character, or character string is known as a *constant*. A constant is an entity that doesn't change whereas a variable is an entity that may change. For example, the number 50 represents a constant integer value. The character string "Programming in C is fun.\n" is an example of a constant character string. C constants can be divided into two major categories:

Primary Constants
Secondary Constants

These constants are further categorized as



Numeric constant
Character constant
String constant

Numeric constant: Numeric constant consists of digits. It required minimum size of 2 bytes and max 4 bytes. It may be positive or negative but by default sign is always positive. No comma or space is allowed within the numeric constant and it must have at least 1 digit. The allowable range for integer constants is -32768 to 32767. Truly speaking the range of an Integer constant depends upon the compiler. For a 16-bit compiler like Turbo C or Turbo C++ the range is –32768 to 32767. For a 32-bit compiler the range would be even greater. Mean by a 16-bit or a 32-bit compiler, what range of an Integer constant has to do with the type of compiler.

It is categorized a **integer constant** and **real constant.** An integer constants are whole number which have no decimal point. Types of integer constants are:

Decimal constant: 0-------9(base 10)
Octal constant: 0-------7(base 8)
Hexa decimal constant: 0----9, A------F(base 16)

In decimal constant first digit should not be zero unlike octal constant first digit must be zero(as 076, 0127) and in hexadecimal constant first two digit should be 0x/ 0X (such as 0x24, 0x87A). By default type of integer constant is integer but if the value of integer constant is exceeds range then value represented by integer type is taken to be unsigned integer or long integer. It can also be explicitly mention integer and unsigned integer type by suffix l/L and u/U.

Real constant is also called floating point constant. To construct real constant we must follow the rule of ,

-real constant must have at least one digit.
-It must have a decimal point.
-It could be either positive or negative.
-Default sign is positive.
-No commas or blanks are allowed within a real constant. Ex.: +325.34
426.0
-32.76

To express small/large real constant exponent(scientific) form is used where number is written in mantissa and exponent form separated by e/E. Exponent can be positive or negative integer but mantissa can be real/integer type, for example $3.6*10^5=3.6e+5$. By default type of floating point constant is double, it can also be explicitly defined it by suffix of f/F.

Character constant

Character constant represented as a single character enclosed within a single quote. These can be single digit, single special symbol or white spaces such as '9','c','$', ' ' etc. Every character constant has a unique integer like value in machine's character code as if machine using ASCII (American standard code for information interchange). Some numeric value associated with each upper and lower case alphabets and decimal integers are as:

A---------- Z ASCII value (65-90)

a----------z ASCII value (97-122)

0----------9 ASCII value (48-59)

; ASCII value (59)

String constant

Set of characters are called string and when sequence of characters are enclosed within a double quote (it may be combination of all kind of symbols) is a string constant. String constant has zero, one or more than one character and at the end of the string null character(\0) is automatically placed by compiler. Some examples are ",sarathina" , "908", "3"," ", "A" etc. In C although same characters are enclosed within single and double quotes it represents different meaning such as "A" and 'A' are different because first one is string attached with null character at the end but second one is character constant with its corresponding ASCII value is 65.

Symbolic constant

Symbolic constant is a name that substitute for a sequence of characters and, characters may be numeric, character or string constant. These constant are generally defined at the beginning of the program as

#define name value , here name generally written in upper case for example

```
#define MAX 10

#define CH 'b'

 #define NAME "sony"
```

Variables

Variable is a data name which is used to store some data value or symbolic names for storing program computations and results. The value of the variable can be change during the execution. The rule for naming the variables is same as the naming identifier. Before used in the program it must be declared. Declaration of variables specify its name, data types and range of the value that variables can store depends upon its data types.

Syntax:

```
  int a;

char c;

float f;
```

Variable initialization

When we assign any initial value to variable during the declaration, is called initialization of variables. When variable is declared but contain undefined value then it is called garbage value. The variable is initialized with the assignment operator such as

```
   Data type variable name=constant;

Example:  int a=20;

          Or int a;

              a=20;
```

statements

Lecture Note: 5

Expressions

An expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical and relational for example:-

```
int z=  x+y    // arithmatic expression

 a>b      //relational

a==b         // logical
func(a, b)   // function call
```

Expressions consisting entirely of constant values are called *constant expressions*. So, the expression
121 + 17 - 110
is a constant expression because each of the terms of the expression is a constant value. But if i were declared to be an integer variable, the expression
180 + 2 – j
would not represent a constant expression.

Operator

This is a symbol use to perform some operation on variables, operands or with the constant. Some operator required 2 operand to perform operation or Some required single operation.

Several operators are there those are, arithmetic operator, assignment, increment , decrement, logical, conditional, comma, size of , bitwise and others.

1. Arithmatic Operator

This operator used for numeric calculation. These are of either Unary arithmetic operator, Binary arithmetic operator. Where Unary arithmetic operator required

only one operand such as +,-, ++, --,!, tiled. And these operators are addition, subtraction, multiplication, division. Binary arithmetic operator on other hand required two operand and its operators are +(addition), -(subtraction), *(multiplication), /(division), %(modulus). But modulus cannot applied with floating point operand as well as there are no exponent operator in c.

Unary (+) and Unary (-) is different from addition and subtraction.

When both the operand are integer then it is called integer arithmetic and the result is always integer. When both the operand are floating point then it is called floating arithmetic and when operand is of integer and floating point then it is called mix type or mixed mode arithmetic . And the result is in float type.

2.Assignment Operator

A value can be stored in a variable with the use of assignment operator. The assignment operator(=) is used in assignment statement and assignment expression. Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression. When variable on the left hand side is occur on the right hand side then we can avoid by writing the compound statement. For example,

```
int x= y;
int  Sum=x+y+z;
```

3.Increment and Decrement

The Unary operator ++, --, is used as increment and decrement which acts upon single operand. Increment operator increases the value of variable by one .Similarly decrement operator decrease the value of the variable by one. And these operator can only used with the variable, but cann't use with expression and constant as ++6 or ++(x+y+z).

It again categories into prefix post fix . In the prefix the value of the variable is incremented 1st, then the new value is used, where as in postfix the operator is written after the operand(such as m++,m--).

EXAMPLE

let y=12;

z= ++y;

y= y+1;

z= y;

Similarly in the postfix increment and decrement operator is used in the operation . And then increment and decrement is perform.

EXAMPLE

let x= 5;

y= x++;

y=x;

x= x+1;

4.Relational Operator

It is use to compared value of two expressions depending on their relation. Expression that contain relational operator is called relational expression.

Here the value is assign according to true or false value.

a.(a>=b) || (b>20)

b.(b>a) && (e>b)

c. 0(b!=7)

5. Conditional Operator

It sometimes called as ternary operator. Since it required three expressions as operand and it is represented as (? , :).

SYNTAX

exp1 ? exp2 :exp3

Here exp1 is first evaluated. It is true then value return will be exp2 . If false then exp3.

EXAMPLE

```
void main()
{
 int a=10, b=2
  int s= (a>b) ? a:b;
  printf("value is:%d");
 }
```

Output:

Value is:10

6. Comma Operator

Comma operator is use to permit different expression to be appear in a situation where only one expression would be used. All the expression are separator by comma and are evaluated from left to right.

EXAMPLE

```
int  i, j, k, l;
for(i=1,j=2;i<=5;j<=10;i++;j++)
```

7. Sizeof Operator

Size of operator is a Unary operator, which gives size of operand in terms of byte that occupied in the memory. An operand may be variable, constant or data type qualifier.

Generally it is used make portable program(program that can be run on different machine) . It determines the length of entities, arrays and structures when their size are not known to the programmer. It is also use to allocate size of memory dynamically during execution of the program.

EXAMPLE

```
main( )

{

int sum;

float f;

printf( "%d%d" ,size of(f), size of (sum) );

printf("%d%d", size of(235 L), size of(A));

}
```

8. Bitwise Operator

Bitwise operator permit programmer to access and manipulate of data at bit level. Various bitwise operator enlisted are

one's complement	(~)
bitwise AND	(&)
bitwise OR	(|)
bitwise XOR	(^)
left shift	(<<)
right shift	(>>)

These operator can operate on integer and character value but not on float and double. In bitwise operator the function showbits( ) function is used to display the binary representation of any integer or character value.

In one's complement all 0 changes to 1 and all 1 changes to 0. In the bitwise OR its value would obtaining by 0 to 2 bits.

As the bitwise OR operator is used to set on a particular bit in a number. Bitwise AND the logical AND.

It operate on 2operands and operands are compared on bit by bit basic. And hence both the operands are of same type.

Logical or Boolean Operator

Operator used with one or more operand and return either value zero (for false) or one (for true). The operand may be constant, variables or expressions. And the expression that combines two or more expressions is termed as logical expression. C has three logical operators :

Operator	Meaning
&&	AND
||	OR
!	NOT

Where logical NOT is a unary operator and other two are binary operator. Logical AND gives result true if both the conditions are true, otherwise result is false. And logial OR gives result false if both the condition false, otherwise result is true.

Precedence and associativity of operators

Operators	Description	Precedence level	Associativity
()	function call	**1**	left to right
[]	array subscript		
→	arrow operator		
.	dot operator		
+	unary plus	2	right to left
-	unary minus		
++	increment		
- -	decrement		
!	logical not		
~	1's complement		
*****	indirection		
&	address		
(data type)	type cast		
sizeof	size in byte		
*****	multiplication	3	left to right
/	division		
%	modulus		
+	addition	4	left to right

-	subtraction		
<<	left shift	**5**	left to right
>>	right shift		
<=	less than equal to	6	left to right
>=	greater than equal to		
<	less than		
>	greater than		
==	equal to	7	left to right
!=	not equal to		
&	bitwise AND	8	left to right
^	bitwise XOR	9	left to right
|	bitwise OR	10	left to right
&&	logical AND	11	
||	logical OR	12	
?:	conditional operator	13	
=, *=, /=, %= &=, ^=, <<= >>=	assignment operator	14	right to left
,	comma operator	15	

Lecture Note: 7

Control Statement

Generally C program statement is executed in a order in which they appear in the program. But sometimes we use decision making condition for execution only a part of program, that is called control statement. Control statement defined