



**GOVERNMENT NATIONAL COLLEGE,  
KARACHI**

## ***Lectures / Notes***

**Subject: Computer Science II**

***Prepared by: Sir Muneer A. Shaikh***

## **Part I – C Language**

## **Chapter # 1**

# **Concepts of Computer Programming**

### **Programming Languages**

- Programming languages are used to create programs that implement specific algorithms.
- Thousands of different programming languages have been created, mainly in the computer field, and many more still are being created every year.
- Each programming language has its own vocabulary and structural rules.
- Programmers use these languages to construct programs containing lists of instructions for the computer to perform specific task.
- Most Popular languages are:
  - BASIC
  - Java
  - Visual Basic
  - C++
  - C# (pronounced C-sharp)
  - PHP
  - SQL
  - JavaScript
- Computers only understand programs in their own machine language. Machine language is the language of 0's and 1's.
- It is difficult to write program in machine language so what is the solution to this dilemma? The answer is in what are called high-level languages i.e. Computer programming languages that look like natural language.
- All programs written in high level language must be translated before their instructions execution.
- Computer languages can be grouped according to which translation process is used to convert the instructions into binary code:
  - Assemblers
  - Interpreters
  - Compilers

## ➤ Generations Of Programming Languages

- First Generation - Machine Language (code)
  - **Machine language** programs were made up of instructions written in binary code.
  - This is the “native” language of the computer.
- Second Generation - Assembly Language
  - **Assembly language** programs are made up of instructions written in mnemonics.
- Third Generation - People-Oriented Programming Languages
  - **High-level languages:** Use statements that resemble English phrases combined with mathematical terms needed to express the problem or task being programmed.
- Fourth Generation - Non-Procedural Languages
  - Programming-like systems aimed at simplifying the programmer’s task of imparting instructions to a computer.
  - Many are associated with specific application packages.
    - ✓ Query Languages:
    - ✓ Report Writers:
    - ✓ Application Generators:
- Fifth Generation - Natural Languages
  - **Object-Oriented Languages:** A language that expresses a computer problem as a series of objects a system contains, the behaviors of those objects, and how the objects interact with each other.





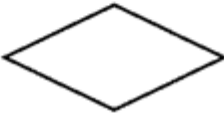
## Lecture # 2

### Flow Chart

- A flow chart, or flow diagram, is a graphical representation of a program that details the sequencing of steps required to create output.
- A typical flow chart uses a set of basic symbols to represent various functions, and shows the sequence and interconnection of functions with lines and arrows.
- Flowcharts are used in analyzing & designing a program.

### Flowchart Symbols

- Flowcharts use special shapes to represent different types of actions or steps in a program.
- Lines and arrows show the sequence of the steps, and the relationships among them.
- Commonly used symbols are:

Name	Symbol	Function
Start/End		Used to markup the starting and ending point
Arrows		Used for connection
Input/Output		Used for input and output information
Process		Used to represent single step
Decision		Used for branching or decision making

### Algorithm

- A programming **algorithm** is a computer procedure that is a lot like a recipe (called a procedure) and tells your computer precisely what steps to take to solve a problem.
- An algorithm is a set of instructions to perform a certain task.
- Most computer programmers spend a large percentage of their time creating algorithms.

**Example:** Write an algorithm and draw a flow chart to calculate the factorial of a number (N).  
Verify your result by a trace table by assuming  $N = 5$ .

**Algorithm:**

Step 1: Input N

Step 2: Factor = 1

Step 3: Counter = 1

Step 4: While (Counter  $\leq$  N)

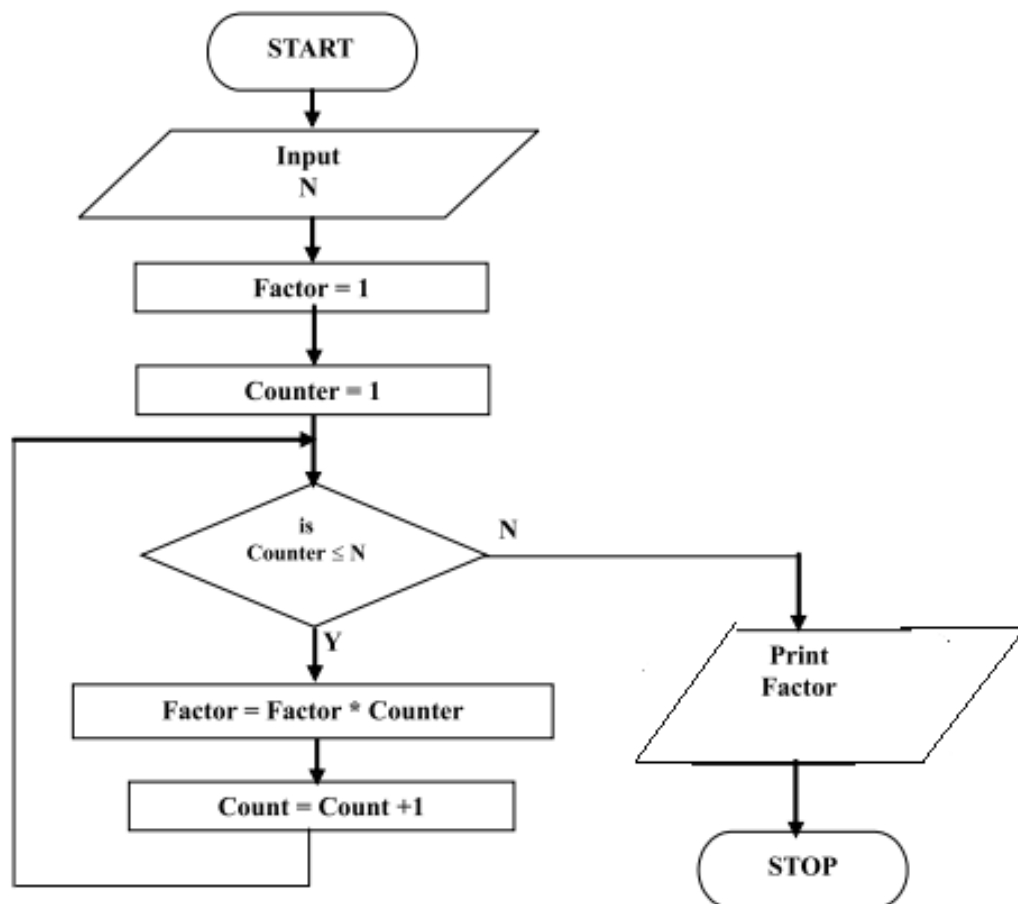
Repeat steps 4 through 6

Step 5: Factor = Factor \* Counter

Step 6: Counter = Counter + 1

Step 7: Print (N, Factor)

**Flow chart**



## Lecture # 3

### High Level Languages:

- Higher level languages make use of English like words and statements and mathematical symbols for instructions.
- Higher level languages make programming easier, since they are relatively easy to learn.
- Less time is required to write programs in high level languages.
- The programmer is not required to know the detailed working of the computer system in order to program in a high level language.
- They are machine independent.
- Higher level languages are also known as problem oriented languages.
- However a high level language is not directly understood by the computer. It is required to be translated into machine language. Therefore they generally execute more slowly and require more memory than the same program written in assembly language.
- The programs which are used to translate programs written in high level language into machine language are known as translators.
- The types of translators are:

**I) Compiler**

**II) Interpreter**

#### **I) Compiler:**

- The compiler translates the entire source program into machine language program at once.
- The source code remains intact.
- Once a program is compiled it can be run as many times as required, without being required to be recompiled.
- A compiler is a program which normally resides on the secondary storage device. It gets loaded into the CPU when the source program is to be translated.
- A compiler checks for errors like illegal symbols, statements etc. during compilation and gives out a list of error messages at the end of execution. This is a very valuable aid to the programmer to correct the programs. However, the compiler is incapable of detecting any logical errors in the program.

## II) Interpreter:

- The interpreter is the program which translates a high level language program into machine language as follows :
  - It takes one statement from the high level language program
  - translates it into a machine instruction and the instruction is immediately executed.
- Since the program is translated statement by statement, the machine level program of the source program is not stored anywhere in memory. Therefore, the program has to be interpreted every time when it has to be run. Thus no object code is generated.
- The interpreted programs are generally slower than compiled programs. However, if any changes are made in the source program it can interpret only those statements and it is not required to compile the entire program again.
- Interpreters are relatively easy to write and smaller in size as compared to compilers.
- Thus assemblers, compilers and interpreters are systems software which translate the source program into object program i.e. program which can be understood by the computer. These translators are also known as language processors.



# Chapter # 2

## Introduction to C Programming Language

### Introduction to C

C is a **programming language**. The **C language** was developed by Dennis Ritchie in 1972 at AT&T Bell Labs. It was called his newly developed language C simply because there was a B programming language already and the **B language** led to the development of **C Language**. **C language** is based on ALGOL and B languages.

Programming language is any language that computer system can understand directly or indirectly to can perform the actions asked by the programmer as set of instructions in form of a computer program. **C language** is a high-level programming language. In the computer world, the further a programming language is from the computer architecture, the higher the language's level. You can imagine that the lowest-level languages are machine languages that computers understand directly. The high-level programming languages, on the other hand, are closer to our human languages. We have to use some special programs, called compilers or interpreters, to translate such a program into a machine-readable code. That is, the text format of all instructions written in a **high-level language** has to be converted into the binary format. The code obtained after the translation is called object code. Prior to the translation, a program in text format is called **source code**.

- C is a general-purpose programming language.
- C is a high-level language that has the advantages of readability, maintainability, and portability.
- C is a very efficient language that allows you to get control of computer hardware and peripherals.
- C is a small language that you can learn easily in a relatively short time.
- Programs written in C can be reused.
- Programs written in C must be compiled and translated into machine-readable code before the computer can execute them.

## C History

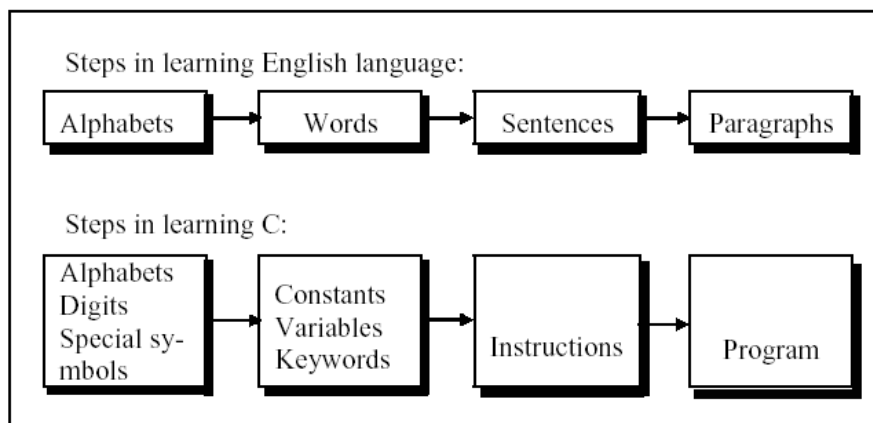
- Developed by Dennis Ritchie at AT&T, early 70s,
- Developed between 1969 and 1973 along with Unix
- Designed for systems programming
  - Operating systems
  - Utility programs
  - Compilers
  - Filters
- Evolved from B, which evolved from BCPL
- Family of languages:
  - BCPL, Martin Richards
  - B (typeless), Ken Thompson, 1970
  - C, Dennis Ritchie, Bell Labs, early 70s
  - C++, Bjarne Stroustrup, Bell Labs, 80s
  - Java, James Gosling Sun, 1995
  - C#, Microsoft, recently

## C vs. C++

- C is a powerful system programming language, and C++ is an excellent general purpose programming language
- C++ is a superset of C
- C++ has all the characteristics of C

## Reasons to learn C

- Main reason is its simplicity reliability, and it's easy to use and easy to learn



## Lecture # 5

### Features of C

- Structured language
  - It has the ability to divide and hide all the information and instruction.
  - Code can be partitioned in C using functions or code block.
- General purpose language
  - Make it ideal language for system programming.
  - It can also be used for business and scientific application.
  - The ability of c is to manipulate bits, byte and addresses.
- Portability
  - Portability is the ability to port or use the software written .
  - One computer C program can be reused.
  - By modification or no modification.
- Code Re-usability & Ability to customize and extend
  - A programmer can easily create his own function
  - It can be used repeatedly in different application
  - C program basically collection of function
  - The function are supported by 'c' library
  - Function can be added to 'c' library continuously
- Limited Number of Key Word
  - There are only 32 keywords in 'C'
  - 27 keywords are given by ritchie
  - 5 is added by ANSI
  - The strength of 'C' is lies in its in-built function
  - Unix system provides as large number of C function
  - Some function are used in operation .
  - Other are for specialized in their application
- Powerful
  - Completeness – business to scientific applications
  - Flexibility – preprocessor, conditional compilation, header files
- Both a high level and low level language
  - High level – symbolic references, user written functions, powerful operator set, derived data types

- Low level – allow reference to storage directly, bits and bit patterns manipulation

## Basic C Program Structure

1. Comments
2. Compiler directives and include files
3. Declarations of global variables and constants
4. Declaration of functions
5. Main function
6. Sub-functions
7. Interrupt service routines

Example: **First C Program**

```
/* My first simple C program */  
#include<stdio.h>  
void main(void)  
{  
    printf("How are you!");  
}
```

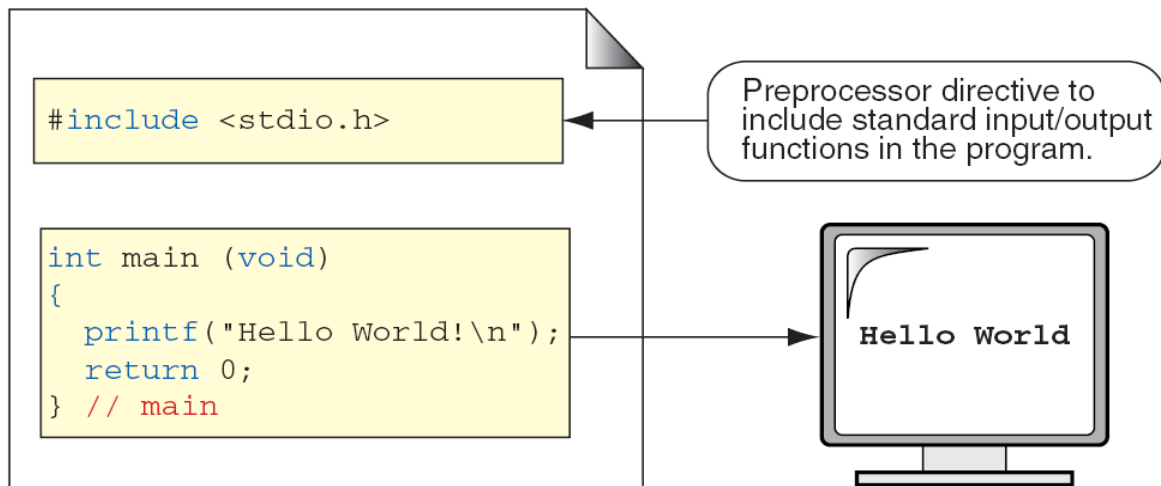


## **Output**

How are you!

Every set of C programs should have one main ( ) function. This is called the driving function. The program execution starts at '{', the opening brace of the main function body and ends at the '}'

closing brace. Every statement in C ends with a ' ; ' ( semicolon ).



## Directives

- Before a C program is compiled, it is first edited by a preprocessor.
- Commands intended for the preprocessor are called directives.
- Example:

```
#include <stdio.h>
```

- <stdio.h> is a **header** containing information about C's standard I/O library.
- Directives always begin with a # character.
- By default, directives are one line long; there's no semicolon or other special marker at the end.

## #include<stdio.h>

- Since we are using a function printf(); in our program in order to print a sentence, so it is necessary to give reference of this function.
- The functionality of printf(); function is defined somewhere in one of many C Standard Libraries, the name of that library is stdio.h.
- Hence including stdio.h at the top of our program become reference for printf function. We also call it prototype, or header file, since it is at the head of our program.
- stdio.h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations.

## Header files

- The files that are specified in the include section is called as header file

- These are precompiled files that has some functions defined in them
- We can call those functions in our program by supplying parameters
- Header file is given an extension .h
- C Source file is given an extension .c

### **void main(void)**

- C language is function oriented language. In this language we write programs through functions.
- C provides some built-in functions. We call it built-in, because functions can also be user defined which we will learn later in this course.
- Hence main is one of the functions of C. Its name is main, because it is an important function, and must be used in a C program, whether you are using other functions or not.
- Void which is written before main means that the function main will not return a value.
- How a value is Returned? You will learn this ahead in your course.
- Void which is written after main means that the function main will not send any argument.
- How arguments are sent/passed? This topic also is covered in coming topics.

### **Main function**

- This is the entry point of a program
- When a file is executed, the start point is the main function
- From main function the flow goes as per the programmers choice.
- There may or may not be other functions written by user in a program
- Main function is compulsory for any c program

**Example: Write a program to display output How are you!**

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
    clrscr();
    printf("How are you!");
    getch();
}
```

### **Explanation:**

- Note that, we have written an extra function: "getch();".

- This function holds the output "How are you" on the output window, till user presses any key. Getch stands for get character.
- Getch() function is the member of conio.h library, hence we include this library at the top of our program.

### **clrscr()**

- Clrscr(); is a function used to clear the previous output from output window.

## Lecture # 6

### Comments in C

- **Comments** (remarks) may appear anywhere within a program, as long as they are placed within the delimiters `/ *` and `*/` (e.g., `/* This is comment */`).
- Such comments are helpful in identifying the program's principal features or in explaining the underlying logic of various program features.
- Single line comment
  - `//` (double slash)
  - Termination of comment is by pressing enter key

- Multi line comment

```
/* ....  
.....*/
```

This can span over to multiple lines

### Some Definitions

- **Integrated Development Environment (IDE):-** It is a software package that makes it possible to edit, compile, link, execute, and debug a program without leaving the environment. It is a screen display with pull down menus. We use menu selections to invoke all the operations necessary to develop our program.
- **Syntax (form):-** The syntax of a language describes the possible combinations of symbols that form a correct program.
- **Compiler:-** It is the part of IDE, that translates our source file into machine language.
- **Keyword:-** Keywords are the words whose meaning has already been explained to the C compiler
- **Source Program:** printable/Readable Program file
- **Object Program:** nonprintable machine readable file
- **Executable Program:** nonprintable executable code
- **Syntax errors:** reported by the compiler
- **Execution/Run-time errors:** reported by the operating system
- **Coding:** writing the instructions in a computer language.
- **Debugging:** detecting and correcting the errors in the programs



## Compilation and Execution

Once you have written the program you need to type it and instruct the machine to execute it. To type your C program you need another program called Editor. Once the program has been typed it needs to be converted to machine language (0s and 1s) before the machine can execute it. To carry out this conversion we need another program called Compiler. Compiler vendors provide an Integrated Development Environment (IDE) which consists of an Editor as well as the Compiler.

There are several such IDEs available in the market targeted towards different operating systems. For example, Turbo C, Turbo C++ and Microsoft C are some of the popular compilers that work under MS-DOS; Visual C++ and Borland C++ are the compilers that work under Windows.

Assuming that you are using a Turbo C or Turbo C++ compiler here are the steps that you need to follow to compile and execute your first C program...

1. Start the compiler at **C>** prompt. The compiler (TC.EXE is usually present in **C:\TC\BIN** directory).
2. Select **New** from the **File** menu.
3. Type the program.
4. Save the program using **F2** under a proper name (say Program1.c).
5. Use **Ctrl + F9** to compile and execute the program.
6. Use **Alt + F5** to view the output.

## Chapter # 3

### C Fundamentals

#### Character Set

- A character denotes any alphabet, digit or special symbol used to represent information.

Following are the valid alphabets, numbers and special symbols allowed in C.

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

#### Identifiers and Keywords

- **Identifiers** are names that are given to various program elements, such as variables, functions and arrays.
- Identifiers consist of letters and digits, in any order, except that *the first character must be a letter*. Both upper- and lowercase letters are permitted, though common usage favors the use of lowercase letters for most types of identifiers.
- Upper- and lowercase letters are not interchangeable (i.e., an uppercase letter is **not** equivalent to the corresponding lowercase letter.)
- The underscore character ( - ) can also be included, and is considered to be a letter.
- An underscore is often used in the middle of an identifier.
- An identifier may also begin with an underscore, though this is rarely done in practice.
- **Example:** The following names are valid identifiers.

X      Y12      sum\_1      \_temperature  
names      area      tax\_rate      TABLE

- The following names are not valid identifiers for the reasons stated.

<b>4th</b>	The first character must be a letter.
<b>x"</b>	illegal characters ( <b>x"</b> ).
<b>order-no</b>	Illegal character (-).
<b>error flag</b>	Illegal character (blank space).

An identifier can be arbitrarily long. Some implementations of C recognize only the first eight characters, though most implementations recognize more (typically, 31 characters).

## **Keywords**

- Keywords are the words whose meaning has already been explained to the C compiler. There are only 32 keywords available in C. The keywords are also called 'Reserved words'.

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

## Lecture # 8

### Data Types

- C supports several different types of data, each of which may be represented differently within the computer's memory.
- Larger the value, larger is the space required in memory to store the number. If you are using a variable whose value varies between 0 and 255 then there is no need to allocate more space in memory for storing this number. The data types are created for efficient use of the memory.
- How much memory is allotted for the various types depends on the machine.
- C has 5 basic built-in data types.
  - character, integer, floating-point, double floating-point, and valueless. These are declared using **char**, **int**, **float**, **double**, and **void**, respectively
- Data type defines a set of values that a variable can store along with a set of operations that can be performed on it.
- Except type **void**, the basic data types may have various modifiers preceding them. A type modifier alters the meaning of the base type to more precisely fit a specific need. The list of modifiers is shown here:
  - signed
  - unsigned
  - long
  - short
- The following Table shows all valid data type combinations supported by C, along with their minimal ranges and typical bit widths..

Type	Typical Size in Bits	Minimal Range
char	8	−127 to 127
unsigned char	8	0 to 255
signed char	8	−127 to 127
int	16 or 32	−32,767 to 32,767
unsigned int	16 or 32	0 to 65,535
signed int	16 or 32	Same as <b>int</b>
short int	16	−32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	Same as <b>short int</b>
long int	32	−2,147,483,647 to 2,147,483,647
long long int	64	−(2 <sup>63</sup> − 1) to 2 <sup>63</sup> − 1 (Added by C99)
signed long int	32	Same as <b>long int</b>
unsigned long int	32	0 to 4,294,967,295
unsigned long long int	64	2 <sup>64</sup> − 1 (Added by C99)
float	32	1E−37 to 1E+37 with six digits of precision
double	64	1E−37 to 1E+37 with ten digits of precision
long double	80	1E−37 to 1E+37 with ten digits of precision

- When a type modifier is used by itself (that is, when it does not precede a basic type), then **int** is assumed. Thus, the following sets of type specifiers are equivalent:

Specifier	Same As
signed	signed int
unsigned	unsigned int
long	long int
short	short int

Although the **int** is implied, it is common practice today to specify the **int** anyway.

## Constants

- Constant is an entity whose value does not change. All the numbers are constants.
- There are four basic types of constants in C. They are *integer constants*, *floating-point constants*, *character constants* and *string constants*

- Integer and floating-point constants represent numbers. They are often referred to collectively as **numeric-type** constants.
- The following rules apply to all numeric-type constants.
  1. Commas and blank spaces cannot be included within the constant.
  2. The constant can be preceded by a minus (-) sign if desired. (Actually the minus sign is an **operator** that changes the sign of a positive constant, though it can be thought of as a part of the constant itself.)
  3. The value of a constant cannot exceed specified minimum and maximum bounds. For each type of constant, these bounds will vary from one C compiler to another.

## Integer Constants

- An **integer constant** is an integer-valued number. Thus it consists of a sequence of digits.
- Integer constants can be written in three different number systems: decimal (base 10), octal (base **8**) and hexadecimal (base 16).
- Beginning programmers rarely, however, use anything other than decimal integer constants.
- A **decimal** integer constant can consist of any combination of digits taken from the set 0 through 9. If the constant contains two or more digits, the first digit must be something other than 0
- **EXAMPLE:** Several valid decimal integer constants are shown below.

**0      1    743      5280    32767    9999**

- The following decimal integer constants are written incorrectly for the reasons stated.

**12,245**                      illegal character (, ).

**36.0**                        illegal character (.).

**10 20 30**                    illegal character (blank space).

**123-45-6789**                illegal character (-).

**0900**                        the first digit cannot be a zero.

## Floating-Point Constants

- A **floating-point constant** is a base- 10 number that contains either a decimal point or an exponent (or both).
- **Example:** Several valid floating-point constants are shown below.

**0                      1.                      0.2                      827.602**

<b>50000.</b>	<b>0.000743</b>	<b>12.3 31</b>	<b>5.0066</b>
<b>2E-8</b>	<b>0.006e-3</b>	<b>1.6667E+8</b>	<b>.12121212e12</b>

- The following are *not* valid floating-point constants for the reasons stated.

1	Either a decimal point or an exponent must be present.
1,000.0	Illegal character (, ).
2E+10.2	The exponent must be an integer quantity
3E 10	Illegal character (blank space) in the exponent.

## Character Constants

- A ***character constant*** is a single character, enclosed in apostrophes (i.e., single quotation marks).
- **Example:** Several character constants are shown below.

**'A'    'X' '3'    '?'    ' '**

Notice that the last constant consists of a blank space, enclosed in apostrophes.

- **Named Constants** - Named constant will have a name like variable but the value of the named constant would not change as

```
float PI=3.142;
```

- Here you declare PI as floating type variable but you are not supposed to change the value of PI.
- However you may accidentally change the value of PI. To avoid this you can use const key word in the declaration as

```
const float PI= 3.142;
```

- Now the compiler itself would not allow you to change the value of PI and you can use PI into any expression as;

```
c=2*PI*j;
```

- **Symbolic or substitution constants** - For using symbolic constant we have to use a preprocessor directive called # define eg.

```
#define PI 3.142
```

- When you compile the C program the preprocessor gets an instruction from the above statement to substitute PI with 3.142. Wherever it finds PI written in the program the same is

substituted by 3.142. PI is not a variable name. PI is not a variable it cannot be used as the variable.

- You can have only one character inside the quotation mark.

```
char ch;  
ch='A ';
```

- Character constants also represent the type **int** . The value then is the ASCII value of the character. eg. ASCII value of **A** is 65, that of **B** is 66 etc. and thus **A** also means 65.

```
char ch;  
ch='A ';  
ch=ch+1;  
printf(“ %c”,ch);
```

will print 'B '

- **Example:** Several character constants and their corresponding values, as defined by the ASCII character set, are shown below.

<u>Constant</u>	<u>Value</u>
'A'	<b>65</b>
'x'	<b>120</b>
'3'	<b>51</b>
'?'	<b>63</b>

Example;

```
#include < stdio.h >  
main ( )  
{  
char a;  
a = 65;  
printf ( " Decimal value of a is %d \n",a );  
printf ( " Character value of a is %c \n",a);  
}
```

**output:**

Decimal value of a is 65

Character value of a is A



## Lecture # 9

### Variables

- As a very basic concept, we can say that a variable is an entity whose value changes. We use a lot of variables in Algebra. When we say

`a = 5 ;`

a is a variable and we are assigning the value of 5 to a.

- Most programs need to a way to store data temporarily during program execution; these storage locations are called *variables*.
- All variables must be declared at top of program, before the first statement.
- The general form of a declaration is  
`type      variable_list;`
- Here, *type* must be a valid data type plus any modifiers, and *variable\_list* may consist of one or more identifier names separated by commas. Here are some declarations:

```
int    i, j, l;
short int    si;
unsigned int    ui;
double  balance, profit, loss;
```

### **Where Variables Are Declared**

- Variables can be declared in three places: inside functions, in the definition of function parameters, and outside of all functions. These positions correspond to local variables, formal parameters, and global variables, respectively.

### Escape Sequences

- Certain nonprinting characters, as well as the backslash (\) and the apostrophe ('), can be expressed in terms of *escape sequences*.
- An escape sequence always begins with a backward slash and is followed by one or more special characters. For example, a line feed (**LF**), which is referred to as a *newline* in C, can be represented as `\n`.
- The commonly used escape sequences are listed below.

Code	Meaning
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	New line

\r	Carriage return
\t	Horizontal tab
\"	Double quote
\'	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert
\?	Question mark

- We have used '\n' character in printf. This is a non printable character and represents new line. Thus when we write

```
printf (" Hello \n");      ( new line character to skip one line . )
printf(" Hi");
```

'Hello' is printed on first line and 'Hi' is printed on the next line.

### **String Constants**

- A *string constant* consists of any number of consecutive characters (including none), enclosed in (double) quotation marks.
- **EXAMPLE:** Several string constants are shown below.

```
"green"    "Washington, D.C. 20005H "    "270-32-3456"
"$19.95"   "THE CORRECT ANSWER is"    "2* ( I+3)/J "
```

### **Constants**

Constant is an entity whose value does not change. All the numbers are constants. However we have to have different types in constant as that of variables for efficient management of memory. Named constant will have a name like variable but the value of the named constant would not change as

```
float PI=3.142;
```

Here you declare PI as floating type variable but you are not supposed to change the value of PI.

However you may accidentally change the value of PI. To avoid this you can use **const** key word in the declaration as

```
const float PI= 3.142;
```

Now the compiler itself would not allow you to change the value of PI and you can use PI into any expression as: `c=2*PI*j;`

## Lecture # 10

### Declarations

- A *declaration* associates a group of variables with a specific data type. All variables must be declared before they can appear in executable statements.
- A declaration consists of a data type, followed by one or more variable names, ending with a semicolon.
- Each array variable must be followed by a pair of square brackets, containing a positive integer which specifies the size (i.e., the number of elements) of the array.
- **Example1:** A C program contains the following type declarations.

```
int    a, b, c;

float  root1, root2;

char   flag , text[80];
```

- Thus, **a**, **b** and **c** are declared to be integer variables, **root1** and **root2** are floating-point variables, **flag** is a char-type variable and **text** is an 80-element, char-type array. Note the square brackets enclosing the size specification for **text**.
- **Example2:** A C program contains the following type declarations.

```
short int  a, b, c;

long int   r, s, t;

int        p, q;

unsigned   x, y;

float      c1, c2, c3;

double     root1, root2;

long float x1, x2;
```

**Example3:** A C program contains the following type declarations.

```
int c = 12;

char star = “*”;

float  sum = 0. ;

double factor = 0.21023e-6;
```

## Expressions

- An **expression** represents a single data item, such as a number or a character.
- The expression may consist of a single entity, such as a constant, a variable, an array element or a reference to a function.
- It may also consist of some combination of such entities, interconnected by one or more **operators**.
- The use of expressions involving operators is particularly common in C, as in most other programming languages.
- Expressions can also represent logical conditions that are either true or false. However, in C the

Conditions **true** and **false** are represented by the integer values **1** and **0**, respectively. Hence logical-type

Expressions really represent numerical quantities.

**Example:** Several simple expressions are shown below.

**a + b**

**x = y**

**c = a + b**

**x <= y**

**x == Y**

**++i**

## Statements

- A **statement** causes the computer to carry out some action.
- There are three different classes of statements in C.
- They are **expression statements**, **compound statements** and **control statements**.
- An expression statement consists of an expression followed by a semicolon.
- The execution of an expression statement causes the expression to be evaluated.
- **Example:** Several expression statements are shown below.

**a = 3;**

**c = a + b ;**

```
++i;  
  
printf("Area = %f", area) ;  
  
;
```

- **Example:** A typical compound statement is shown below.

```
{  
  
pi = 3.141593;  
  
circumference = 2. * pi * radius;  
  
area = pi * radius * radius;  
  
}
```

## Lecture # 11

### Assignment

- A variable can be given a value by means of *assignment*:

```
height = 8;
```

The number 8 is said to be a *constant*.

- Before a variable can be assigned a value—or used in any other way—it must first be declared.
- A constant assigned to a float variable usually contains a decimal point:

```
profit = 2150.48;
```

- It's best to append the letter f to a floating-point constant if it is assigned to a float variable:

```
profit = 2150.48f;
```

Failing to include the f may cause a warning from the compiler.

- An int variable is normally assigned a value of type int, and a float variable is normally assigned a value of type float.
- Mixing types (such as assigning an int value to a float variable or assigning a float value to an int variable) is possible but not always safe.
- Once a variable has been assigned a value, it can be used to help compute the value of another variable:

```
height = 8;
```

```
length = 12;
```

```
width = 10;
```

```
volume = height * length * width;
```

```
/* volume is now 960 */
```

# Chapters 4

## Operators and Expressions

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

The data items that operators act upon are called *operands*. Some operators require two operands, while others act upon only one operand. A few operators permit only single variables as operands.

### Arithmetic Operators

- There are five *arithmetic operators* in C. They are

<u>Operator</u>	<u>Purpose</u>
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder after integer division

The % operator is sometimes referred to as the **modulus operator**.

Example: Suppose that a and b are integer variables whose values are 10 and 3, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values.

<u>Expression</u>	<u>Value</u>
a + b	13
a - b	7
a * b	30
a / b	3
a % b	1

Now suppose that v1 and v2 are floating-point variables whose values are 12.5 and 2.0, respectively.

Several arithmetic expressions involving these variables are shown below, together with their resulting values.

<u>Expression</u>	<u>Value</u>
$v1 + v2$	14.5
$v1 - v2$	10.5
$v1 * v2$	25.0
$v1 / v2$	6.25

Finally, suppose that `c1` and `c2` are character-type variables that represent the characters `P` and `T`, respectively. Several arithmetic expressions that make use of these variables are shown below, together with their resulting values.

<u>Expression</u>	<u>Value</u>
<code>c1</code>	80
<code>c1 + c2</code>	164
<code>c1 + c2 + 5</code>	169
<code>c1 + c2 + '5'</code>	217

Note that `P` is encoded as (decimal) 80, `T` is encoded as 84, and `5` is encoded as 53 in the ASCII character set.

If one or both operands represent negative values, then the addition, subtraction, multiplication and division operations will result in values whose signs are determined by the usual rules of algebra. Integer division will result in truncation toward zero; i.e., the resultant will always be smaller in magnitude than the true quotient.

The interpretation of the remainder operation is unclear when one of the operands is negative. Most versions of C assign the sign of the first operand to the remainder. Thus, the condition

$$a = ((a / b) * b) + (a \% b)$$

will always be satisfied, regardless of the signs of the values represented by `a` and `b`.

Example: Suppose that `a` and `b` are integer variables whose values are 11 and -3, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values.

<u>Expression</u>	<u>Value</u>
<code>a + b</code>	8



$a - b$	14
$a * b$	-33
$a / b$	-3
$a \% b$	2

**Example:** Let **r1** and **r2** be floating-point variables whose assigned values are -0.66 and **4.50**. Several arithmetic expressions involving these variables are shown below, together with their resulting values.

<u>Expression</u>	<u>Value</u>
$r1 + r2$	3.84
$r1 - r2$	-5.16
$r1 * r2$	-2.97
$r1 / r2$	-0.1466667

Example: Suppose that **i** is an integer variable whose value is 7, **f** is a floating-point variable whose value is 5.5, and **c** is a character-type variable that represents the character **w**. Several expressions which include the use of these variables are shown below. Each expression involves operands of two different types.

Assume that the ASCII character set is being used.

<u>Expression</u>	<u>Value</u>	<u>Type</u>
$i + f$	12.5	double-precision
$i + c$	126	Integer
$i + c - '0'$	78	integer
$(i + c) - (2 * f / 5)$	123.8	double-precision

## Lecture # 13

### Rules of operator precedence:

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
*, /, or %	Multiplication, Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

### Unary Operators

C includes a class of operators that act upon a single operand to produce a new value. Such operators are known as **unary operators**. Unary operators usually precede their single operands, though some unary operators are written after their operands.

Perhaps the most common unary operation is **unary minus**, where a numerical constant, variable or expression is preceded by a minus sign.

Example: Here are several examples which illustrate the use of the unary minus operation.

-743            -0X7FFF            -0.2            -5E-8  
-root1            -( x + Y)            -3 \* (x + y)

### **Increment & decrement operators**

There are two other commonly used unary operators: The **increment operator**, ++, and the **decrement operator**, --. The increment operator causes its operand to be increased by 1, whereas the decrement operator causes its operand to be decreased by 1. The operand used with each of these operators must be a single variable.

**Example:** Suppose that i is an integer variable that has been assigned a value of 5. The expression ++i, which is equivalent to writing i = i + 1, causes the value of i to be increased to 6. Similarly, the expression --i, which is equivalent to i = i - 1, causes the (original) value of i to be decreased to 4.

The increment and decrement operators can each be utilized two different ways, depending on whether the operator is written before or after the operand. If the operator precedes the operand (e.g., ++i), then the operand will be altered in value **before** it is utilized for its intended purpose within the program. If, however, the operator **follows** the operand (e.g., i++), then the value of the operand will

be altered *after* it is utilized.

**Example:** Suppose that  $x$  and  $y$  are integer variables whose values are 10 and 20, respectively. The value of the expression  $-x + y$  will be  $-10 + 20 = 10$ . Note that the unary minus operation is carried out before the addition. Now suppose that parentheses are introduced, so that the expression becomes  $-(10 + 20)$ . The value of this expression is  $-(10 + 20) = -30$ . Note that the addition now precedes the unary minus operation.

## Assignment Operators

- An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	$a = b$	$a = b$
+=	$a += b$	$a = a + b$
-=	$a -= b$	$a = a - b$
*=	$a *= b$	$a = a * b$
/=	$a /= b$	$a = a / b$
%=	$a \% = b$	$a = a \% b$

## Lecture # 14

### Relational and Logical Operators

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

- These six operators are used to form logical expressions, which represent conditions that are either true or false.
- EXAMPLE: Suppose that i, j and k are integer variables whose values are 1, 2 and 3, respectively. Several logical expressions involving these variables are shown below.

Expression	Interpretation	Value
i < j	true	1
(1 + j) >= k	true	1
(j + k) > (i + 5)	false	0
k != 3	false	0

### Logical Operators:

- The logical operators act upon operands that are themselves logical expressions. The net effect is to combine the individual logical expressions into more complex conditions that are either true or false. The result of a logical **and** operation will be true only if both operands are true, whereas the result of a logical **or** operation will be true if either operand is true or if both operands are true. In other words, the result of a logical **or** operation will be false only if both operands are false.

<u>Operator</u>	<u>Meaning</u>
&&	and
	or
!	not

EXAMPLE: Suppose that i is an integer variable whose value is 7, f is a floating-point variable whose value is 5.5, and c is a character variable that represents the character 'w'. Several complex logical expressions that make use of these variables are shown below.

<b>Expression</b>	<b>Interpretation</b>	<b>Value</b>
(i >= 6) && (c == ' w ')	true	1
(i >= 6)    (c == 119)	true	1
(f < 11) && (i > 100)	false	0
(c != ' p ' )    ((i + f ) <= 10)	true	1

The hierarchy of operator precedence covering all of the operators discussed so far has become extensive. These operator precedences are summarized below, from highest to lowest.

<u>Operator category</u>	<u>Operators</u>
unary operators	- ++ -- ! sizeof
arithmetic multiply, divide and remainder	* / %
arithmetic add and subtract	+ -
relational operators	< <= > >=
equality operators	== !=
logical <i>and</i>	&&
logical <i>or</i>	

# Chapters 5

## Input & Output

- When we say **Input**, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.
- When we say **Output**, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.
- **Types of I/O Statements**
  1. Formatted I/O
    - scanf()
    - printf()
  2. Unformatted I/O
    - gets(), puts()
    - getch(), getche(), putch()
    - getchar(), putchar()
- The functions used for input and output are stored in the header file stdio.h. If programmer use any of the above function it is necessary to include header file.

### 1. Formatted Input & Output

#### (i) scanf() function

- The function used to get input from the user during execution of program and stored in a variable of specified form is called scanf() function.

▪ **Syntax:**     scanf("format string",& variable name);

#### **Types:**

- Single input e.g scanf("%d",&a);
- Multiple input scanf("%d %c %d", &a,&op,&b);|

#### ▪ **Example**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int a;
    clrscr();
    printf("Enter integer value:");
```

```
scanf("%d",&a);
printf("The value you enter = %d",a);
getch();
}
```

**Output** Enter integer value:5 The value you enter =5

## (ii) printf ()Function

- This function is used to display text, constant or value of variable on screen in specified format.
- **Syntax:**    printf(“format string”, argument list);
- **Types:**
  - printf(“hello world”);    // Printf()with no argument list
  - printf(“the value of integar=%d”,a); //Printf() with one argument
  - printf(“Sum of %d+%d=%d”,a,b,a+b);    //Printf()with more than one argument
- **Example:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
printf(“HELLO WORLD!”);
getch();
}
```

**Output** HELLO WORLD!

## Formatted Output with printf

- Format Conversion Specifiers:
  - d -- displays a decimal (base 10) integer
  - l -- used with other specifiers to indicate a "long"
  - e -- displays a floating point value in exponential notation
  - f -- displays a floating point value
  - g -- displays a number in either "e" or "f" format
  - c -- displays a single character
  - s -- displays a string of characters

## Lecture # 16

### 2. Unformatted Input & Output

#### (i) gets()

- This special function is used to input string values from the user during execution of the program. As the user press enter key when typing after the string. A null character (\0) is automatically entered at the end of the string.
- Syntax:      gets(variable name);

#### (ii) puts function

- The function used to display the string value on the screen.
- Syntax: puts(parameter/value/variable);
- **Example:**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    char name[15];
    clrscr();
    printf("enter your name:");
    gets(name);
    printf("your name is:");
    puts(name);
    getch();
}
```

**Output:** enter your name: somebody your name is: somebody

#### (iii) getch()

- getch() is used to get a character from console but does not echo to the screen. It reads a single character directly from the keyboard, without echoing to the screen.
- Syntax:

```
char ch;
ch = getch();
```

- **Program**

```
#include <stdio.h>
void main()
{
    char ch;
    ch = getch();
    printf("Input Char Is :%c",ch);
}
```

#### (iv) getche()

- getche() is used to get a character from console, and echoes to the screen.



- getch reads a single character from the keyboard and echoes it to the current text window, using direct video or BIOS.

- **Syntax**

```
char ch;
ch = getch();
```

- **Program**

```
#include <stdio.h>
void main()
{
    char ch;
    ch = getch();
    printf("Input Char Is :%c",ch);
}
```

#### (v) **putch()**

- putch displays any alphanumeric characters to the standard output device. It displays only one character at a time.

- **Syntax**

```
putch(variable_name);
```

- **Example**

```
char ch = 'a';
putch(ch)
```

#### (vi) **getchar()**

- The getchar function is a part of the standard C input/output library. It returns a single character from a standard input device (typically a keyboard). The function does not require any arguments, though a pair of empty parentheses must follow the word getchar.

- **Syntax**

```
character variable = getchar( );
```

- where character variable refers to some previously declared character variable.

#### (vii) **putchar()**

- The putchar function like getchar is a part of the standard C input/output library. It transmits a single character to the standard output device (the computer screen).

- **Syntax**

```
putchar(character variable)
```

- where character variable refers to some previously declared character variable.

- **Program**

```
#include <stdio.h>
int main( )
{
    int c;
    printf( "Enter a value :");
```

```
c = getchar( );  
printf( "\nYou entered: ");  
putchar( c );  
return 0;  
}
```

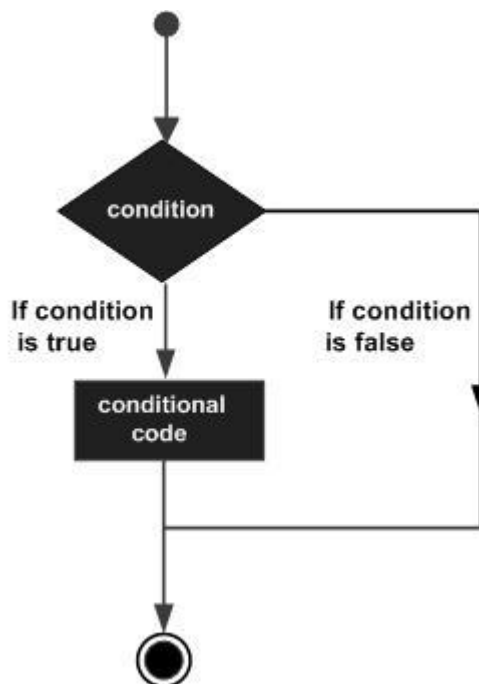
Enter a value : this is test  
You entered: t

## Chapters 6

### Decision Statements

#### Decision Making

- Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- The general form of a typical decision making structure found in most of the programming languages is:



- C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.
- C programming language provides the following types of decision making statements.

#### Control Structure

- A statement that is used to control the flow of execution in a program is called control structure. It combines instruction into logical unit. Logical unit has one entry point and one exit point.
- **Types of control structures**
  1. Sequence
  2. Selection (branching)

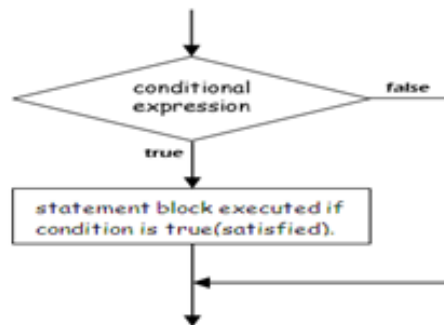
### 3. Repetition (Iteration)

#### Selection Statements

- Selection structures allow the computer to make decisions in your programs. It selects a statement or set of statements to execute on the basis of a condition.
- **Types:**
  1. if-else statement
  2. switch statement

##### (i) If Statement

- The simplest if structure involves a single executable statement.
- Execution of the statement occurs only if the condition is true.
- **Syntax:**     if (condition)  
                  statement;
- **Flow chart:**



- **Example:**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int marks;
    clrscr();
    printf("Enter your marks:");
    scanf("%d",&marks);
    if(marks >=50)
    printf("CONGRATULATIONS...!!! you have passed.");
    getch();
}
```

#### Output

```
Enter your marks: 67
CONGRATULATIONS...!!! you have passed.
```

## Lecture # 18

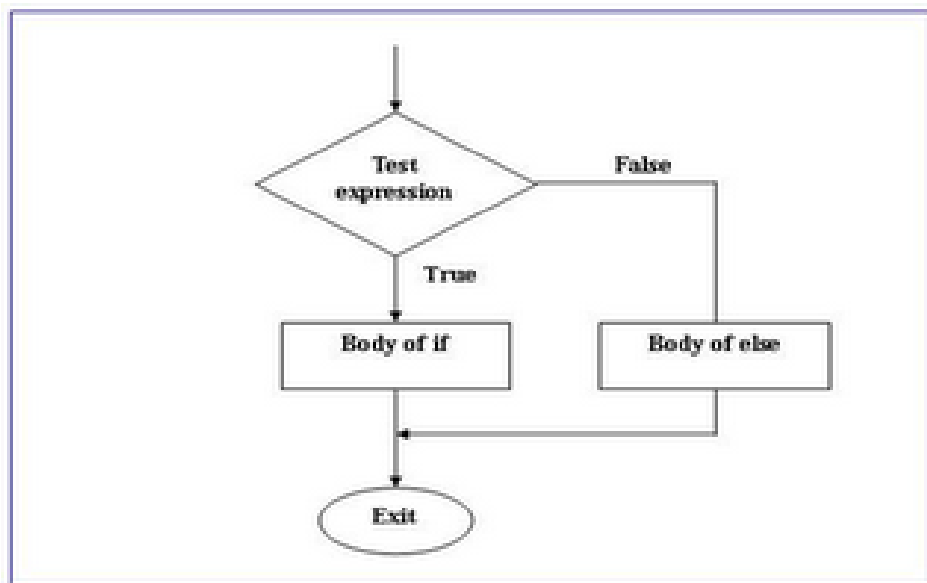
### (ii) If-else statement

- In if-else statement, if the condition is true, then the true statement(s), immediately following the if-statement are executed otherwise the false statement(s) are executed. The use of else basically allows an alternative set of statements to be executed if the condition is false.

- **Syntax:**

```
if (condition)
{
    Statement(s);
}
else
{
    Statement(s);
}
```

- **Flow chart:**



- **Example:**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int y;
    clrscr();
    printf("Enter a year:");
```

```
scanf("%d",&y);
if (y % 4==0)
printf("%d is a leap year.",y);
else
printf("%d is not a leap year.",y);
getch();
}
```

### Output

Enter a year: 2004

2004 is a leap year.

### (iii) if -else if statement (else if ladder)

- It can be used to choose one block of statements from many blocks of statements. The condition which is true only its block of statements is executed and remaining are skipped.

- **Syntax:**

```
if (condition)
{
statement(s);
}
else if (condition)
{
statement(s);
}
else
{
statement;
}
```

- **Example:**

```
#include <stdio.h>
#include <conio.h>
void main()
{
int num;
clrscr();
printf("Enter your mark: ");
scanf("%d",&num);
printf(" You entered %d", num);
if(num >= 80)
{
printf(" You got A grade");
}
else if ( num >=60 && num<80)
{
printf(" You got B grade");
}
```

```

else if ( num >=50&& num<60)
{
printf(" You got C grade");
}
else if ( num < 50)
{
printf(" You Failed in this exam");
}
getch();
}

```

### Output

Enter your mark: 67

You got B grade

### (iv) Nested if

- In nested-if statement if the first if condition is true the control will enter inner if. If this is true the statement will execute otherwise control will come out of the inner if and the else statement will be executed.

- **Syntax:**

```

if ( condition 1)
{
    if ( condition 2)
        statement 1 ;
    else
        statement 2 ;
}
else
{
    if (condition 3)
        statement 3;
    else
        statement 4;
}

```

- **Example:**

```

#include<stdio.h>
#include<conio.h>
void main ()
{
int a,b,c;
clrscr();
printf("Enter three number:");
scanf("%d %d %d",&a,&b,&c);
if (a>b)
{
    if (a>c)
        printf("a is big");
    else

```

```
        printf("c is big");
    }
    else
    {
        if(b>c)
            printf("b is big");
        else
            printf("c is big");
    }
    getch();
}
```

**Output**

Enter three number:

4

7

6

b is big



## Lecture # 19

### (v) Switch Statement

- The control statements which allow us to make a decision from the number of choices is called switch (or) Switch-case statement.
- It is a multi way decision statement, it test the given variable (or) expression against a list of case value.
- **Rules for Switch**
  - The expression in the switch statement must be an integer or character constant.
  - No real numbers are used in an expression.
  - The default is optional and can be placed anywhere, but usually placed at end.
  - The case keyword must be terminated with colon (:);
  - No two case constant are identical.
  - The values of switch expression are compared with case constant in the order specified i.e from top to bottom.
  - A switch may occur within another switch, but it is rarely done. Such statements are called as nested switch statements.
- The switch statement is very useful while writing menu driven programs.
- **Syntax:**

```
switch(expression)
{
case 1:
statement;
break;
case 2:
statement;
break;
.
.
.
.

case N:
statement;
break;
default:
statement;
}
```

### ▪ Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
int num1,num2,ans1,choice;

float ans2;

clrscr();
printf(" \n enter two numbers");
scanf("%d%d",&num1,&num2);
printf(" \n1.Addition \n 2.Subtraction \n 3.Multiplication \n 4.Division \n");
printf("\n Enter the choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
ans1=num1+num2;
printf("Addition =%d",ans1);
break;
case 2:
ans1=num1-num2;
printf("Subtraction =%d",ans1);
break;
case 3:
ans1=num1*num2;
printf("Multiplication =%d",ans1);
break;
case 4:
ans2=(float)num1/num2;
printf("Division =%f",ans2);
break;
default:
printf("wrong choice");
break;
}
getch();
}
```

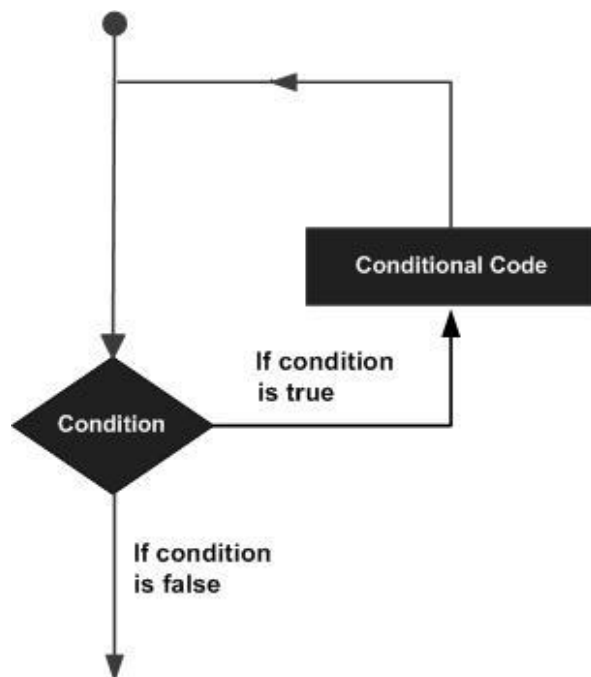
### Output

```
enter two numbers
23
41
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter the choice 1
64
```

## Lecture # 20

### Chapters 7 Loop Control Statements

- You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages:



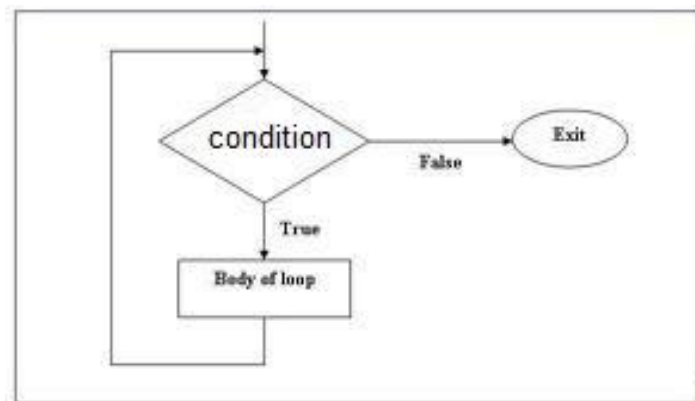
- C programming language provides the following types of loops to handle looping requirements.
- **Types**
  1. While loop
  2. Do while loop
  3. For loop

#### while loop

- Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
- **Syntax**  
initialization;

```
while (condition)
{
    statement;
    increment/decrement;
}
```

- **Flow chart**



- **Example**

```
#include<stdio.h>
#include<conio.h>
void main (void)
{
    int n=1;
    clrscr();
    while (n<=5)
    {
        printf("\n %d",n);
        n++;
    }
    getch();
}
```

**Output**

```
1
2
3
4
5
```

## Lecture # 21

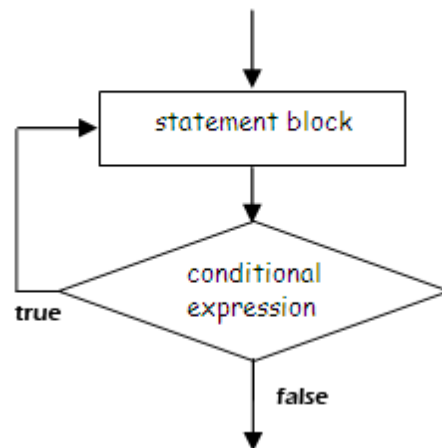
### (ii) do while loop

- Do while loops are useful where loop is to be executed at least once.
- In do while loop condition comes after the body of loop.
- This loop executes one or more statements while the given condition is true.

- **Syntax**

```
initialization;  
do  
{  
    statement(s);  
    increment/decrement;  
}while (condition);
```

- **Flow chart**



### Example:

```
#include<stdio.h>  
#include<conio.h>  
void main (void)  
{  
    int a=1;  
    clrscr();  
    do  
    {  
        printf("\n %d",a);  
        a++;  
    }  
    while (a<=5);  
    getch();  
}
```

## Output

1  
2  
3  
4  
5

## Difference Between While Loop and Do – While Loop

S.No.	while loop	do-while loop
1.	The while loop tests the condition before each iteration.	The do – while loop tests the condition after the first iteration.
2.	If the condition fails initially the loop is Skipped entirely even in the first iteration.	Even if the condition fails initially the loop is executed once.

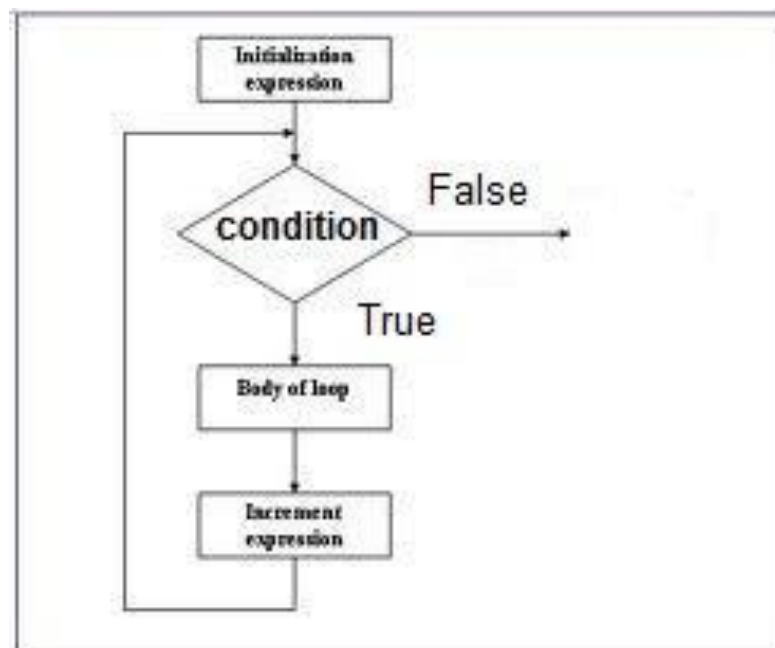
### (iii) For loop

- For loops are used when the number of iterations is known before entering the loop. It is also known as counter-controlled loop.

- Syntax**

```
for (initialization; condition; increment/decrement)
{
    statement(s);
}
```

- Flow chart**



**Example:**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int n;
    for(n=1;n<=5;n++)
    {
        printf("\n %d",n);
    }
    getch();
}
```

**Output**

1  
2  
3  
4  
5

## Lecture # 22

### Nested loop

- A loop within a loop is called nested loop.
- In this the outer loop is used for counting rows and the internal loop is used for counting columns.
- Any loop can be used as inner loop of another loop.
- **Syntax**

```
for (initialization; condition; increment/decrement)
{
    for(initialization; condition, increment/decrement)
    {
        statements(s);
    }
}
```

- **Example**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int i,j;
    clrscr();
    for(i=1;i<=3;i++)
    {
        for(j=1;j<=2;j++)
        {
            printf("\n%d\t%d",i,j);
        }
    }
    getch();
}
```

#### Output

```
1  1
1  2
2  1
2  2
3  1
3  2
```



## Loop Control Statements

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- C supports the following control statements

### 1. break statement

- Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
- The general form of the break statement is

break;

#### **Program**

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i;
    clrscr();
    i = 1;
    for(i=0;i<5;i++)
    {
        if(i==3)
            break;
        printf("%d",i);
    }
}
```

#### **Output**

0  
1  
2

### 2. continue statement

- Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- The general form or the syntax of the continue statement is

continue;

#### **Program**

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i;
    clrscr();
```

```

        i = 1;
        for(i=0;i<5;i++)
        {
            if(i==3)
                continue;
            printf("%d",i);
        }
        getch();
    }

```

**Output**

```

0
1
2
4

```

### 3. goto statement

- It is an unconditional transfer of control. It transfers the control to the specific point.
- The goto statement is marked by label statement.
- Label statement can be used anywhere in the function above or below the goto statement.
- It is written as goto label;

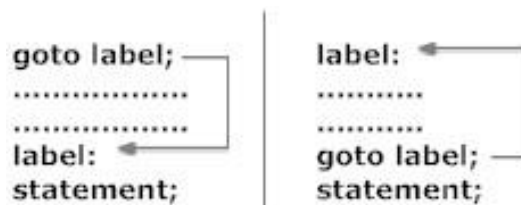


Figure: Working of goto statement

**Example:**

```

#include <stdio.h>
#include<conio.h>
void main()
{
    int a=3;
    clrscr();
    Start: printf("Welcome\n");
    if(a<5)
    {
        goto Start;
        a++;
    }
    getch();
}

```

**Output**

```

Welcome
Welcome

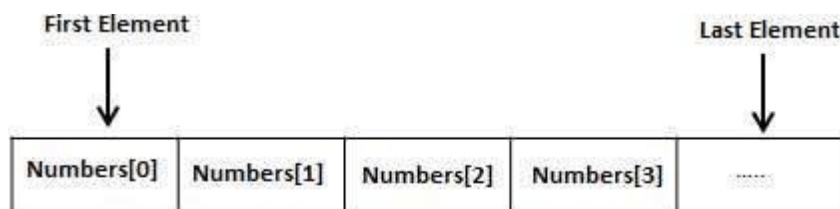
```

## Chapters 8

### Arrays

#### Array

- C programming language provides a data structure called **the array**, which can store a fixed-size sequential collection of elements of the same type.
- Array is a collection of homogenous data stored under unique name. The values in an array are called as 'elements of an array.' These elements are accessed by numbers called as 'subscripts or index numbers.'
- Arrays may be of any variable type.
- Array is also called as 'subscripted variable.'
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.
- A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.



- **Types of an Array:**
  - i. One / Single Dimensional Array
  - ii. Two Dimensional Array

#### Single / One Dimensional Array:

- The array which is used to represent and store data in a linear form is called as 'single or one dimensional array.'
- **Syntax:**

<Data-type> <array\_name> [size];

- **Example:**

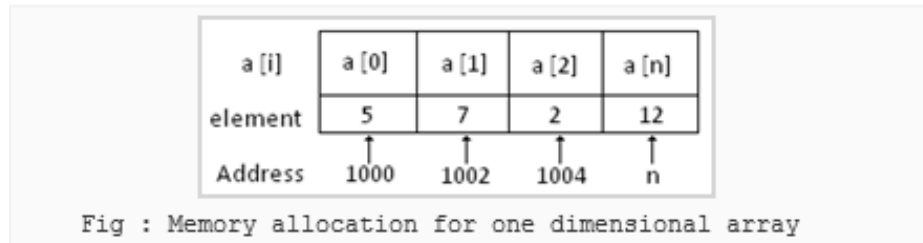
```
int a[3] = {2, 3, 5};  
char ch[20] = "Techno Exam" ;
```

float stax[3] = {5003.23, 1940.32, 123.20} ;

▪ **Total Size (in Bytes):**

- Total size = length of array \* size of data type
- In above example, a is an array of type integer which has storage size of 3 elements. The total size would be  $3 * 2 = 6$  bytes.

\* MEMORY ALLOCATION :



**Features:**

- Array size should be positive number only.
- String array always terminates with null character ('\0').
- Array elements are countered from 0 to n-1.
- Useful for multiple reading of elements (numbers).

**Disadvantages:**

- There is no easy method to initialize large number of array elements.
- It is difficult to initialize selected elements

**Program**

```
/* C program to find the sum marks of n students using arrays */
#include <stdio.h>
void main()
{
    int marks[10],i,n,sum=0;
    printf("Enter number of students: ");
    scanf("%d",&n);
    for(i=0;i<n;++i)
    {
        printf("Enter marks of student%d: ",i+1);
        scanf("%d",&marks[i]);
        sum+=marks[i];
    }
    printf("Sum= %d",sum);
}
```

**Output**

```
Enter number of students: 3
Enter marks of student1: 12
Enter marks of student2: 31
Enter marks of student3: 2
sum=45
```

## Lecture # 24

### Multidimensional Arrays

- C programming language allows programmer to create arrays of arrays known as multidimensional arrays.
- For example: Here, *a* is an array of two dimension, which is an example of multidimensional array.

	col 1	col 2	col 3	col 4	col 5	col 6
row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]

Figure: Multidimensional Arrays

#### Initialization of Multidimensional Arrays

- In C, multidimensional arrays can be initialized in different number of ways.  
int c[2][3]={ { 1,3,0}, {-1,5,9} };  
OR  
int c[][3]={ { 1,3,0}, {-1,5,9} };  
OR  
int c[2][3]={ 1,3,0,-1,5,9};

#### Initialization Of three-dimensional Array

```
double cprogram[3][2][4]={  
    { {-0.1, 0.22, 0.3, 4.3}, {2.3, 4.7, -0.9, 2} },  
    { {0.9, 3.6, 4.5, 4}, {1.2, 2.4, 0.22, -1} },  
    { {8.2, 3.12, 34.2, 0.1}, {2.1, 3.2, 4.3, -2.0} }  
};
```

Suppose there is a multidimensional array arr[i][j][k][m]. Then this array can hold i\*j\*k\*m numbers of data. Similarly, the array of any dimension can be initialized in C programming.

#### Program

```
#include<stdio.h>  
  
void main()  
{  
    int a[5][5],b[5][5],c[5][5];  
    int row1,row2,col1,col2,i,j,k;  
    printf("Enter the row value of first matrix\n");  
    scanf("%d",&row1);  
    printf("Enter the column value of first matrix\n");  
    scanf("%d",&col1);  
    printf("Enter the row value of second matrix\n");  
    scanf("%d",&row2);  
    printf("Enter the column value of second matrix\n");  
    scanf("%d",&col2);  
    if((row1==row2)&&(col1==col2))
```

```

{
printf("Matrix can be added \n");
printf("Enter the values of first matrix\n");
for(i=1;i<=row1;i++)
{
for(j=1;j<=col1;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("Enter the values of second matrix\n");
for(i=1;i<=row2;i++)
{
for(j=1;j<=col2;j++)
{
scanf("%d",&b[i][j]);
}
}
for(i=1;i<=row1;i++)
{
for(j=1;j<=col1;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
}
printf("Sum of the two matrix is\n");
for(i=1;i<=row1;i++)
{
for(j=1;j<=col1;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
}
else
printf("Addition cannot be perform");

```

}

### **OUTPUT**

Enter the row value of first matrix 2

Enter the column value of first matrix 2

Enter the row value of second matrix 2

Enter the column value of second matrix 2

Matrix can be added

Enter the values of first matrix

2 4

5 6

Enter the values of second matrix

3 1

2 8

Sum of the two matrix

5 5

7 14

## Chapters 9

### Strings

#### Strings

- Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.
- The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

- If you follow the rule of array initialization then you can write the above statement as follows

```
char greeting[] = "Hello";
```

- Following is the memory presentation of the above defined string in C/C++

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

- Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string –

```
#include <stdio.h>
int main () {
    char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
    printf("Greeting message: %s\n", greeting );
    return 0;
}
```

- When the above code is compiled and executed, it produces the following result:

Greeting message: Hello

- C supports a wide range of functions that manipulate null-terminated strings:



## Function & Purpose

### **strcpy(s1, s2);**

Copies string s2 into string s

### **strcat(s1, s2);**

Concatenates string s2 onto the end of string s1

### **strlen(s1);**

Returns the length of string s1

### **strcmp(s1, s2);**

Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2

### **strchr(s1, ch);**

Returns a pointer to the first occurrence of character ch in string s1

### **strstr(s1, s2);**

Returns a pointer to the first occurrence of string s2 in string s1

- The following example uses some of the above-mentioned functions

```
#include <stdio.h>
#include <string.h>
int main () {
    char str1[12] = "Hello";
    char str2[12] = "World";
    char str3[12];
    int len ;
    /* copy str1 into str3 */
    strcpy(str3, str1);
    printf("strcpy( str3, str1) :  %s\n", str3 );
    /* concatenates str1 and str2 */
    strcat( str1, str2);
    printf("strcat( str1, str2):  %s\n", str1 );
    len = strlen(str1);
    printf("strlen(str1) :  %d\n", len );
    return 0;
}
```

- When the above code is compiled and executed, it produces the following result

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):  HelloWorld
```

## Chapters 10

### Functions & Data Files

#### Function

- A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.
- You can divide up your code into separate functions.
- A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.
- The C standard library provides numerous built-in functions that your program can call.
- A function can also be referred as a method or a sub-routine or a procedure, etc.

#### Function Characteristics

- The use of programmer-defined functions allows a large program to be broken down into a number of smaller, self-contained components, each of which has some unique, identifiable purpose.
- Functions allow a particular group of instructions be accessed repeatedly, from several different places within the program.
- The use of functions also enables a programmer to build a *customized library* of frequently used routines.

#### Function Prototypes

- Function prototypes are usually written at the beginning of a program, ahead of any programmer-defined functions (including main).
- The general form of a function prototype is  
$$\text{data-type name( type1 arg1 , type2 arg2, . . . , type n arg n ) ;}$$
- where data- type represents the data type of the item that is returned by the function, name represents the function name, and type 1, type 2, . . . , type n represent the data types of the arguments arg 1 , arg 2, . . . , arg n.
- Notice that a function prototype resembles the first line of a function definition.

#### Defining a Function

- The general form of a function definition in C programming language is as follows:

```

return_type  function_name( parameter list )
{
    body of the function
}

```

- A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function:

- **Return Type** – A function may return a value. The **return\_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword **void**.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** – The function body contains a collection of statements that define what the function does.

### Example

- Given below is the source code for a function called **max()**. This function takes two parameters num1 and num2 and returns the maximum value between the two –

```

/* function returning the max between two numbers */
int max(int num1, int num2) {

    /* local variable declaration */
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}

```

## Lecture # 27

### Function Declarations

- A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A function declaration has the following parts:

```
return_type function_name( parameter list );
```

- For the above defined function max(), the function declaration is as follows:

```
int max(int num1, int num2);
```

### Calling a Function

- To use a function, you will have to call that function to perform the defined task.
- When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.
- To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example

```
#include <stdio.h>

/* function declaration */

int max(int num1, int num2);

int main () {

    /* local variable definition */

    int a = 100;

    int b = 200;

    int ret;

    /* calling a function to get max value */

    ret = max(a, b);

    printf( "Max value is : %d\n", ret );
```

```

        return 0;
    }

    /* function returning the max between two numbers */

    int max(int num1, int num2) {

        /* local variable declaration */

        int result;

        if (num1 > num2)

            result = num1;

        else

            result = num2;

        return result;

    }

```

- We have kept max() along with main() and compiled the source code. While running the final executable, it would produce the following result:

Max value is : 200

### **Function Arguments**

- If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.
- Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.
- While calling a function, there are two ways in which arguments can be passed to a function:

#### 2. **Call by value**

- This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

#### 3. **Call by reference**

- This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

## Lecture # 28

### Library Functions

- The C language is accompanied by a number of *library functions* that carry out various commonly used operations or calculations.
- A typical set of library functions will include a fairly large number of functions that are common to most C compilers, such as those shown in Table below.

<i>Function</i>	<i>Type</i>	<i>Purpose</i>
<code>abs(i)</code>	int	Return the absolute value of <i>i</i> .
<code>ceil(d)</code>	double	Round up to the next integer value (the smallest integer that is greater than or equal to <i>d</i> ).
<code>cos(d)</code>	double	Return the cosine of <i>d</i> .
<code>cosh(d)</code>	double	Return the hyperbolic cosine of <i>d</i> .
<code>exp(d)</code>	double	Raise <i>e</i> to the power <i>d</i> ( $e = 2.7182818 \dots$ is the base of the natural (Naperian) system of logarithms).
<code>fabs(d)</code>	double	Return the absolute value of <i>d</i> .
<code>floor(d)</code>	double	Round down to the next integer value (the largest integer that does not exceed <i>d</i> ).
<code>fmod(d1,d2)</code>	double	Return the remainder (i.e., the noninteger part of the quotient) of <i>d1/d2</i> , with same sign as <i>d1</i> .
<code>getchar()</code>	int	Enter a character from the standard input device.
<code>log(d)</code>	double	Return the natural logarithm of <i>d</i> .
<code>pow(d1,d2)</code>	double	Return <i>d1</i> raised to the <i>d2</i> power.
<code>printf(...)</code>	int	Send data items to the standard output device (arguments are complicated – see Chap. 4).
<code>putchar(c)</code>	int	Send a character to the standard output device.
<code>rand()</code>	int	Return a random positive integer.
<code>sin(d)</code>	double	Return the sine of <i>d</i> .
<code>sqrt(d)</code>	double	Return the square root of <i>d</i> .
<code>srand(u)</code>	void	Initialize the random number generator.
<code>scanf(...)</code>	int	Enter data items from the standard input device (arguments are complicated – see Chap. 4).
<code>tan(d)</code>	double	Return the tangent of <i>d</i> .
<code>toascii(c)</code>	int	Convert value of argument to ASCII.
<code>tolower(c)</code>	int	Convert letter to lowercase.
<code>toupper(c)</code>	int	Convert letter to uppercase.

## Lecture # 29

### Data Files:

- Data files allow us to store information permanently, and to access and alter that information whenever necessary.
- In C, an extensive set of library functions is available for creating and processing data files

### Opening and Closing a Data File

- When working with a data file, the first step is to establish a **buffer area**, where information is temporarily stored while being transferred between the computer's memory and the data file.
- This buffer area allows information to be read from or written to the data file more rapidly than would otherwise be possible. The buffer area is established by writing

FILE *\*ptvar*;

- Where FILE (uppercase letters required) is a special structure type that establishes the buffer area and *ptvar* is a pointer variable that indicates the beginning of the buffer area.
- A data file must be **opened** before it can be created or processed. It also specifies how the data file will be utilized, i.e., **as** a read-only file, a write-only file, or a read write file, in which both operations are permitted.
- The library function fopen is used to open a file. This function is typically written as

*ptvar* = fopen(*file-name*, *file-type*);

Where *file-name* and *file-type* are strings that represent the name of the data file and the manner in which the data file will be utilized.

- The name chosen for the *file-name* must be consistent with the rules for naming files, as determined by the computer's operating system. The *file-type* must be one of the strings shown in Table.

**File-Type Specifications**

<i>File-Type</i>	<i>Meaning</i>
'r'	Open an existing file for reading only.
'w'	Open a new file for writing only. If a file with the specified <i>file-name</i> currently exists, it will be destroyed and a new file created in its place.
'a'	Open an existing file for appending (i.e., for adding new information at the end of the file). A new file will be created if the file with the specified <i>file-name</i> does not exist.

- Finally, a data file must be closed at the end of the program. This can be accomplished with the library function fclose. The syntax is simply:

fclose (*ptvar*);

## C Programs

### 1. Write a program to calculate area of circle (get radius from user)

```
#include<stdio.h>
#include<conio.h>
main()
{
    float r,Area;
    printf("Enter radius");
    scanf("%f",&r);
    Area=3.14*r*r;
    printf("Area of circle = %f",Area);
    getch();
}
```

### 2. Write a program to calculate circumference of circle (get radius from user)

```
#include<stdio.h>
#include<conio.h>
main()
{
    float r,c;
    printf("Enter radius");
    scanf("%f",&r);
    c=2*3.14*r;
    printf("Circumference of circle = %f",c);
    getch();
}
```

### 3. Write a Program that gets a character from user and displays its ASCII code

```
#include<stdio.h>
#include<conio.h>
main()
{
    char ch;
    printf("Enter a character");
    scanf("%c",&ch);
    printf("ASCII code =%d",ch);
    getch();
}
```



**4. Write a program that gets two numbers from user and perform all arithmetic operations.**

```
#include<stdio.h>
#include<conio.h>
main()
{
float a,b;
printf("Enter two numbers");
scanf(" %f %f ", &a , &b );
printf("%f + %f = %f \n",a,b ,a+b);
printf("%f - %f = %f \n",a,b ,a-b);
printf("%f * %f = %f \n",a,b ,a*b);
printf("%f / %f = %f \n",a,b ,a/b);
getch();
}
```

**5. Write a Program that gets distance in miles from user and convert it into kilometers**

```
#include<stdio.h>
#include<conio.h>
main()
{
float m, km;
printf("Enter distance in miles");
scanf("%f",&m);
km=m * 1.6;
printf(" %f miles = %f km", m ,km);
getch();
}
```

**6. Write a Program that gets temperature in foreignheite from user and convert it into Celsius**

```
#include<stdio.h>
#include<conio.h>
main()
{
float f,c;
printf("Enter temperature in foreignheite");
scanf("%f",&f);
c=5/9.0 *(f-32);
printf("%f foreinheite = % f centigrade ",f,c);
getch();
}
```

**7. Write a program that takes a number from user and calculates its factorial with while loop**

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
clrscr();
int num=0;
int fact=1;
printf("plz enter a number \n");
scanf("%d", &num);
while(num>0)
{
fact=fact*num;
Num--;
}
printf("factorial of the number is %d",fact);
getch();
}
```

**8. Write a program that takes a number from user and calculates its factorial with for loop.**

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
clrscr();
int num=0;
int fact=1;
printf("plz enter a number \n");
scanf("%d", &num);
for(int i=num;i>0;i--)
{
fact=fact*i;
}
printf("factorial of the number is %d",fact);
getch();
}
```

- 9. Write a program that takes a number form user and decides whether the number is even or odd.**

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
clrscr();
int num=0;
printf("plz enter a number \n");
scanf("%d", &num);
if(num%2==0)
{
printf("number is even \n");
}
else
{
printf("number is odd \n");
}
getch();
}
```

- 10. Write a program that takes obtained marks and total marks from user and calculates the average and assigns the grade.**

```
#include<stdio.h>
#include<conio.h>
void main(void )
{
clrscr();
int obt_marks=0;
int total_marks=0;
float avg=0.0;
printf("plz enter your obtained marks \n");
scanf("%d",&obt_marks);
printf("plz enter total marks \n");
scanf("%d",&total_marks);
avg=(obt_marks*100.0)/total_marks;
if(avg>=80)
{
printf("Grade A+");
}
else if(avg>=70 && avg<80)
```

```

{
printf("Grade A");
}
else if(avg>=60 && avg<70)
{
printf("Grade B");
}
else if(avg>=50 && avg<60)
{
printf("Grade C");
}
else if(avg>=40 && avg<50)
{
printf("Grade D");
}
else
{
printf("Grade F");
}
getch();
}

```

**11. Write a program that takes a number from user and print its table with the help of for loop.**

```

#include<stdio.h>
#include<conio.h>
main()
{
int table=0;
int prod=0;
printf("plz enter a number \n");
scanf("%d",&table);
for(int i=0;i<=10;i++)
{
prod=table*i;
printf(" %d * %d =%d \n ",table,i,prod);
}
getch();
}

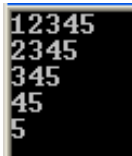
```

**12. Write a program that displays the following shape by using nested for loop**

```
* * * * *
* * * * *
* * * * *
* * * * *
```

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
clrscr();
for(int i=1;i<=4;i++)
{
for(int j=1;j<=5;j++)
{
printf("*");
}
printf("\n");
}
getch();
}
```

**13. Write a program that displays the following shape by using nested for loop**



```
12345
2345
345
45
5
```

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
clrscr();
for(int i=1;i<=5;i++)
{
for(int j=i;j<=5;j++)
{
printf("%d",j);
}
printf("\n");
}
getch();
}
```

**14. Write a program that gets length and width of a rectangle from user, passes it to a function that finds Area and returns it to main that displays the area.**

```
#include<stdio.h>
#include<conio.h>
float Area(float l,float w)
{
float A= l*w;
return A;
}
void main(void)
{
float l,w;
clrscr();
printf("plz enter length and width of rectangle");
scanf("%f %f",&l,&w);
float area =Area(l,w);
printf("Area of rectangle = %f",area);
getch();
}
```

**15. Write a program that gets a number from user passes it to function that finds and displays its table.**

```
#include<stdio.h>
#include<conio.h>
void table(int n)
{
for(int i=1;i<=10;i++)
{
printf("%d * %d= %d \n", n , i , n*i);
}
}
void main(void)
{
int n;
printf("Enter a number \n");
scanf("%d",&n);
table(n);
getch();
}
```

**16. Write a program that prints the square of all the numbers from 1 to 10.**

```
#include<conio.h>
#include<stdio.h>
void main(void)
{
    int i;
    clrscr();
    printf("Number\tSquare\n");
    for(i=1;i<=10;i++)
    {
        printf("%d\t%d\n",i,i*i);
    }
    getch();
}
```

**17. Write a program that reads temperature and prints a message as given below.**

<u>Temperature</u>	<u>Message</u>
t>35	It is hot!
t≥20, t≤35	Nice Day!
T<20	It is cold!

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
    int temp;
    clrscr();
    printf("Enter the temperature");
    scanf("%d",&temp);
    if(temp>35)
        printf("It is hot!");
    if(temp>=20 && temp<=35)
        printf("Nice day");
    if(temp<20)
        printf("It is cold!");
    getch();
}
```

**18. Write a program that reads three numbers and prints the largest.**

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
    int a,b,c;
    clrscr();
    printf("Enter any three no.s and get the largest no.");
    scanf("%d %d %d",&a,&b,&c);
    if(a>b && a>c)
        printf("%d is the largest no ",a);
    else
        if(b>a && b>c)
            printf("%d is the largest no",b);
        else
            if(c>a && c>b)
                printf("%d is the largest no",c);
            else
                printf("All no. are equal");
    getch();
}
```

**19. Write a program using a function to calculate the area of a rectangle.**

```
#include<stdio.h>
#include<conio.h>
int area(int,int);
void main(void)
{
    int height,width,ans;
    clrscr();
    printf("Enter height of the rectangle\n");
    scanf("%d",&height);
    printf("Enter the width of the rectangle\n");
    scanf("%d",&width);
    ans=area(height,width);
    printf("The area of the rectangle is %d",ans);
    getch();
}
int area(int a,int b)
{
    int c;
    c=a*b;
    return c;
}
```



### **Important Questions:**

#### **Q 1. What is computer program?**

**Ans.** The set of instructions given to the computer to solve a specific problem is called computer program. Computer can solve problems with the help of computer program. Computer programs are written in programming languages.

#### **Q 2. What is programming language?**

**Ans.** Programming language is used to communicate with computer. All computer programs are written in programming languages. Every programming language has a set of alphabets and rules. The instructions of computer program are written by using the alphabets and rules defined by the programming language.

#### **Q 3. List different types of programming language.**

**Ans.** There are two types of programming languages.

1. Low level languages
2. High level languages

#### **Q 4. What is high level language?**

**Ans.** A programming language that is close to human language is called high level language. The instructions written in high level language look like English language sentences. High level languages are easy to learn and understand.

#### **Q 5. What is low level language?**

**Ans.** A language that is close to the language of computer. Computer itself uses this language is called a low level language. There are two types of low level languages.

1. Machine language
2. Assembly language

#### **Q 6. What is machine language?**

**Ans.** Machine language is also called binary language. There are only two alphabets of machine language those are zero (0) and one (1). Computer can understand only machine language. The programs written in other programming languages are first translated to machine language, and then used on computer. It is called the native language of computer.

**Q 7. What is assembly language?**

**Ans.** It is a programming language in which machine language instructions are replaced by English like words. These words are known as mnemonics. It is pronounced as Ne-Monic. An assembly language used English like words. It is easy to write a program in assembly language. It is mostly used for system software.

**Q 7. What is source program?**

**Ans.** A computer program written in high level language is called source program. Source program is also called source code.

**Q 8. What is object program?**

**Ans.** A computer program in machine language is called object program. Object program is also called object code. Machine language is native language of computer so object program directly runs on computer.

**Q 9. What is language translator?**

**Ans.** A language processor is software that converts a program written in any programming language into machine language. Every language has its own processor. Language processor is also called language translator.

**Q 10. What is compiler?**

**Ans.** Software that converts a high level language program into machine language is called compiler. Every language has its own compiler. First a program is written in high level language. Then it is given to compiler. Compiler detects and tells about errors in programs. When high level language program is error free compiler converts it into machine language.

**Q 11. What is interpreter?**

**Ans.** It is a program that converts a source program into object program one statement at a time. It takes one statement of source program, translates it in machine code and executes it. After the execution of one statement, it takes next statement and repeats this process till the last instruction of program.

**Q 12. What is assembler?**

**Ans.** It is a language translator that converts a program written in assembly language into machine language.

**Q 13. What is structured programming language?**

**Ans.** In structured programming language the program is divided into parts or modules. These modules are combined to make a complete program. It is easy to write and debug a program written in structured programming language. There is less chance of error.

**Q 14. What is unstructured programming language?**

**Ans.** In unstructured programming language whole program consists of a single unit. It does not consist of parts or modules. It is difficult to write and debug a program written in unstructured programming language. There is more chance of error.

**Q 15. What is preprocessor?**

**Ans.** A compiler is a program that translates a high level language program into machine language. This process is called compilation. Preprocessor is a program that modifies a C program before compilation. Preprocessor directives are instructions for the preprocessor.

**Q 16. What is preprocessor directive?**

**Ans.** Preprocessor directives are instructions for the preprocessor. These instructions are written at the beginning of the program. Every preprocessor directive starts with # symbol. After # symbol "include" or "define" directives are used. Preprocessor directives are also called compiler directives.

**Q 17. What is the work of include directive?**

**Ans.** Include is a preprocessor directive. It is used to include header files in to the program. The relevant header file of the library function, we want to use in programs is mentioned at the beginning of program by using include directive.

**Q 18. What is the work of define directive?**

**Ans.** Define preprocessor directive is used to define a constant macro. Its general syntax is:

**#define Macro-Name expression**

**Q 19. What is statement terminator?**

**Ans.** Every C language statement ends with semicolon ";". Semicolon at the end of statement is called statement terminator.

**Q 20. What are delimiters?**

**Ans.** Curly braces at the beginning and end of the main function are called delimiters. C language statements are written between delimiters.

**Q 21. What is main function?**

**Ans.** Every C language program must contain a main() function. A program without main function cannot be executed. Instructions of programs are written between the curly braces {} of main() function. These statements enclosed in main() function are called body of the main() function.

**Q 22. What are bugs and debugging?**

**Ans.** While writing a program the programmer may come across many errors. The error in a program is called bug. The process of finding and removing errors is called debugging.

**Q 23. What is meant by creating a program?**

**Ans.** Writing source code statements is called creating C program. Turbo C IDE can be used to create and edit program. First open Turbo C IDE. Then select new from file menu. A new edit window will be opened. The cursor blinks in the window. Cursor control keys can be used for cursor movements. We write the program statements in the window and save it as a program file.

**Q 24. What is meant by editing a program?**

**Ans.** Writing and editing source program is the first step. Source code is written in C language according to the type of the problem, in any text editor. Changing source code and removing errors in the program is called editing a program.

**Q 25. What is meant by compiling a program?**

**Ans.** Computer does not understand C language. It understands only machine language. So C language code is converted into machine language. The process of converting source code into machine code is called compiling. Compiler is a program that compiles source code. If compiling is successful source program is converted into object program. Object program is saved on disk. The extension of file is ".obj".

**Q 26. What is meant by linking a program?**

**Ans.** The process of combining required library functions with object program is called linking. This is done with the help of a program called linker. It is a part of compiler. The linker combines object

program produced by compiler and library function. It produces and saves a final machine language file. This file is called executable file. The extension of executable file is ".exe".

**Q 27. What is meant by executing a program?**

**Ans.** The process of running an executable file is called executing. After linking C program can be executed. A program loader is used to transfer executable file from secondary storage to main memory. The program can be executed by selecting run from run menu bar of Turbo C IDE or by pressing Ctrl + F9 keys from keyboard.

**Q 28. List name of some high level language.**

**Ans.** High level languages:

- C
- C++
- C#
- COBOL
- BASIC
- FORTRAN
- PASCAL
- JAVA

**Q 29. What is Turbo C++?**

**Ans.** Turbo C++ is an integrated development environment (IDE) for creating C and C++ programs. Borland international has developed it. It is also called TC editor. It is used to create, edit and save programs. It also has powerful debugging features. These help us in finding and removing errors from a program. We can easily compile program. Linking a program is also very easy. It is also used to execute a program.

**Q 30. What are necessary step to prepare a C program?**

**Ans.** Step to prepare a C program

- Creating & Editing & Saving
- Compiling & Linking
- Loading & Running

**Q 31. What are header files?**

**Ans.** Header files are part of C compiler. C language provides many built-in programs. Every program has a unique name. These programs are called built-in functions or library functions. Every library function can perform a specific task. We can use these library functions in our C language program. These functions are divided into groups according to their functionality. A group of same type of functions is stored in a same file. This file is called header file.

**Q 32. What is C statement?**

**Ans.** Every instruction written in C language program is called a C statement. Every statement ends with a semicolon ";". Semicolon is called statement terminator.

**Q 33. What are syntax errors?**

**Ans.** The rule for writing a program in a specific programming language is called syntax of the language. We must follow the syntax of a language. Syntax error occurs when the statements of program are not according to syntax. Compiler detects syntax errors. If there is a syntax error in program, It cannot be compiled successfully. Compiler tells about the location and type of syntax error. Syntax errors can be removed easily.

**Q 34. What are logical errors?**

**Ans.** The error that is due to wrong algorithm is called logical error. These errors occur due to the wrong logic of program. Compiler cannot detect these errors. A program having logical errors gives wrong results on execution. These errors are difficult to find, as compiler cannot detect these errors. The programmer should examine the whole program to find logical errors.

**Q 35. What are runtime errors?**

**Ans.** These errors occur during the execution of program are called runtime errors. When runtime error occur the execution of program stops and computer shows an error message. These errors occur when program wants to perform such task that computer cannot perform.

**Q 36. What is ANSI C?**

**Ans.** C language is very powerful and flexible language. Wide range of application programs are written in C language. American National Standard Institute (ANSI) made standard version of C language in late 1980s. This standard version of C is also called ANSI C. New version of C has many new features that were not available in older versions.

**Q 37. List any four advantages of C language?**

**Ans.** Advantages of C language

- Easy to learn
- Easy to Remove Errors
- Machine Independence
- Standard Syntax
- Shorter Programs

**Q 38. What is meant by machine independence?**

**Ans.** A low level language program can run only on the type of computers for which it is written. So low level languages are machine dependent. A program written in high level language is machine independent. It can run on all types of computers.

**Q 39. What is the difference between compiler and interpreter?**

**Ans.** Software that converts a high level language program into machine language is called compiler. Every language has its own compiler. Compiler detects and tells about errors in programs. When high level language program is error free compiler converts it into machine language. So compiler is software that converts a source program into object program as a whole. Object code is used for execution.

It is a program that converts a source program into object program one statement at a time. It takes one statement of source program, translates it in machine code and executes it. Each time we execute a program by using its source code.

**Q 40. What is Algorithms?**

**Ans.** An algorithm refers to the step by step instructions written to solve any problem.

**Q 41. What is Flowchart?**

**Ans.** A flowchart is a diagrammatic or symbolic representation of an algorithm. It uses various symbols to represent the operations to be performed.

**Q 42. Name the four basic data types in “C” language?**

**Ans.** The four basic data types in “c” language are as follows

1. char – a character

2. int – an integer, in the range -32,767 to 32,767
3. long int – a larger integer (up to +-2,147,483,647)
4. float – a floating-point number
5. double – a floating-point number, with more precision and perhaps greater range than float

**Q 43. Describe at least four different format specifiers?**

**Ans. The common four format specifiers are:**

1. %d: -An integer whole number
2. %f: -a floating point number
3. %c: -a single character
4. %s: -a string of value of characters.

**Q 44. Define and explain scanf () function?**

**Ans.** The Scanf () function can be used to get input into a program and it requires two arguments. First a format specifier defines the type of data to be entered, then the name of the variable in which the input will be stored. This scanf () function is responsible for giving input into the program.

**Q 44. Define and explain printf () function?**

**Ans.** The printf() function is used to display/output values of variable in the monitor. The printf function has general form: printf (“format specifiers”,variables)

**Q 45. What are the maximum and minimum possible ranges of values for long and short type?**

**Ans.** If the int variable is created by default as a ‘long’ type it typically will have a possible range of values from a maximum of +214748347 and a minimum of -2147483648. For ‘short’ type these are the maximum and minimum values +327676 and minimum -32768.

**Q 46. What exactly is a ‘variable scope’, ‘local variables’ and ‘global variables’?**

**Ans.** The extent to which a variable is accessible in a program is called the ‘variable scope’. Variables declared internally inside a function are known as ‘local’ variables. Variables declared externally outside a function are known as ‘global’ variables.



**Q 47. What are signed values?**

**Ans.** When an int variable is declared it can by default contain either positive or negative integer values. These are known as 'signed' values. The range of positive values is determined by your system.

**Q 48. Define reserved words.**

**Ans.** C programs are constructed from a set of reserved words which provide control and from libraries which perform special functions. The basic instructions are built up using a reserved set of words, such as main, for, if, while, default, double, extern, for, and int, to name just a few.

**Q 49. What is the purpose of type declaration in C?**

**Ans.** All variables used in a C program are declared using the appropriate data types to enable the compiler to allocate the required number of bytes in RAM to store values of these variables in memory.

**50. ) What is identifier?**

**Ans.** An identifier is a name used to identify a variable, function, symbolic constant and so on.

**Q 51. Mention the different types of operators used in C?**

**Ans.** The different types of operators used in C are:

1. Arithmetic operator
2. Relational operators
3. Logical Operators
4. Increment and decrements operators
5. Assignment operators
6. Conditional operator
7. Bitwise operators

**Q 52. What is a Loop control statement?**

**Ans.** Loop control structures are used to execute and repeat a block of statements depending on the value of a condition. There are 3 types of loop control statements in C

1. for loop
2. while loop
3. do – while loop

**Q 53. Explain while loop?**

**Ans.** A while loop has one control expression, and it is executed as long as that expression is true. The general syntax of a while loop is

```
while( expression )  
  
{  
  
statements  
  
}
```

We use a while loop when a statement or group of statements which may have to be executed a number of times to complete their task. The controlling expression represents the condition

**Q 54. Explain for loop?**

**Ans.** A for loop is used to execute and repeat a block of statements depending on a condition. The syntax of a for loop is

```
for( ; ; )  
  
{  
  
statements  
  
}
```

**Q 55. List a few unconditional control statements in C.**

**Ans.** Few unconditional control statements in C are:

1. break statement
2. continue statement
3. goto statement
4. exit() function

**Q 56. What is an array?**

**Ans.** An array is a collection of values of the same data type. Values in array are accessed using array name with subscripts in brackets[]. Syntax of array declaration is

```
data type array_ name[size];
```

**Q 57. Define function?**

**Ans.** A function is a module or block of program code which deals with a particular task. Each function has a name or identifier by which is used to refer to it in a program. A function can accept a number of parameters or values which pass information from outside, and consists of a number of statements and declarations, enclosed by curly braces { }, which make up the doing part of the object

**Q 58. Differentiate built-in functions and user – defined functions.**

**Ans.** Built – in functions are used to perform standard operations such as finding the square root of a number, absolute value and so on. These are available along with the C compiler and are included in a program using the header files math.h, string.h and so on.

User defined functions are written by the user or programmer to compute a value or perform a task. It contains a statement block which is executed during the runtime whenever it is called by the main program.

**Q 59. ) Distinguish between actual and formal arguments.**

**Ans.** Actual arguments are variables whose values are supplied to the function in any function call. Formal arguments are variables used to receive the values of actual arguments from the calling program.

**Q 60. What is recursion?**

**Ans.** A function calling itself again and again to compute a value is referred to as recursive function or recursion. Recursion is useful for branching processes and is effective where terms are generated successively to compute a value.

**Q 61. What are Library functions?**

**Ans.** Library functions are built in programs available along with the compiler which perform some standard mathematical operations.

**Q 62. What is an operator and operand?**

**Ans.** An operator performs an operation like addition, subtraction and so on and produces a value. Variables and constants upon which operations are performed are called operands.

# **Part II - MS Access**

## Lecture # 30

### Introduction to Microsoft Access:

- Microsoft Access is a computer application (database software) used to create and manage computer-based databases on desktop computers and/or on connected computers (a network).
- Microsoft Access can be used for personal information management (PIM), in a small business to organize and manage data, or in an enterprise to communicate with servers.
- MS Access is a relational database, meaning that data is stored in multiple tables that are related to each other.
- With Microsoft Access, you can analyze large amounts of data faster and more efficiently than with Excel or other types of spreadsheets.
- It has many built in features to assist you in constructing and viewing your information.

### Data:

- Raw facts such as an employee's name, address, salary and CNIC number
- Data is the coded representation of information for use in a computer.
- Data has attributes, such as type and length. Data may be:
  - Numeric
  - Alphabetic
  - Alpha-numeric
  - Graphic

### Database

- A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.
- A database may be generated and maintained manually or it may be computerized. The library card catalog is an example of a database that may be created and maintained manually. A computerized database may be created and maintained either by a group of application programs written specially for that task or by a database management system (DBMS).
- ▶ Popular Databases Management Systems are: Oracle, Microsoft SQL Server, MySQL,

### Advantages of Database

- Better support for client/server systems
- Flexible data sharing

- Controlled redundancy
- Better security

### **Database Applications:**

- ▶ Banking: all transactions
- ▶ Airlines: reservations, schedules
- ▶ Universities: registration, grades
- ▶ Sales: customers, products, purchases
- ▶ Online retailers: order tracking, customized recommendations
- ▶ Manufacturing: production, inventory, orders, supply chain
- ▶ Human resources: employee records, salaries, tax deductions

## Lecture # 31

### Objects of Microsoft Access:

- Within Access there are four major objects:
  - Forms - Forms allow you to enter data and to view data stored in your tables
  - Tables - Tables store your data in your database
  - Reports - Reports allow you to print data based on queries/tables that you have created
  - Queries - Queries ask questions about information stored in your tables

### Forms

- A form is nothing more than a graphical representation of a table.
- Forms let you enter and display specific data in a customized format
- You can add, update, and delete records in your table by using a form.
- If you change a record in a form, it will be changed in the table also.
- A form is very good to use when you have numerous fields in a table. This way you can see all the fields in one screen.
- Basic Types of Forms:
  - Single Record & Datasheet

### Tables:

A table is a two-dimensional list of items so that the items are arranged by categories. A complete or incomplete series of items that represent each category is called a **record (Tuple)**. Therefore, a table can be represented as follows:

The diagram illustrates a table structure. At the top, a box labeled "Categories of Information" has four arrows pointing down to the column headers of a table: "Name", "Email Address", "Phone Number", and "Relationship". To the left of the table, a box labeled "Records" has four arrows pointing right to the rows of the table. The table contains four rows of data.

	Name	Email Address	Phone Number	Relationship
	Bill	bill@yahoo.com		Friend
	James	jamesemail.com	(102) 399-2893	
	Hermine		(101) 447-8384	Cousin
	Khan	@Khan.com		

In database development, a category is represented as a column. Sometimes it is also called a **field (Attribute)**. A record is represented as a row.

## **Reports**

- Reports display and print formatted data
- A report is an effective way to present your data in a printed format. Because you have control over the size and appearance of everything on a report, you can display the information the way you want to see it.

## **Queries**

- Queries select and modify specific data
- “Queries convert data to information”
- They are used to populate forms and reports
- MS Access uses a visual query wizard to help beginner (and advanced) users construct queries



## Lecture # 32

### Database File:

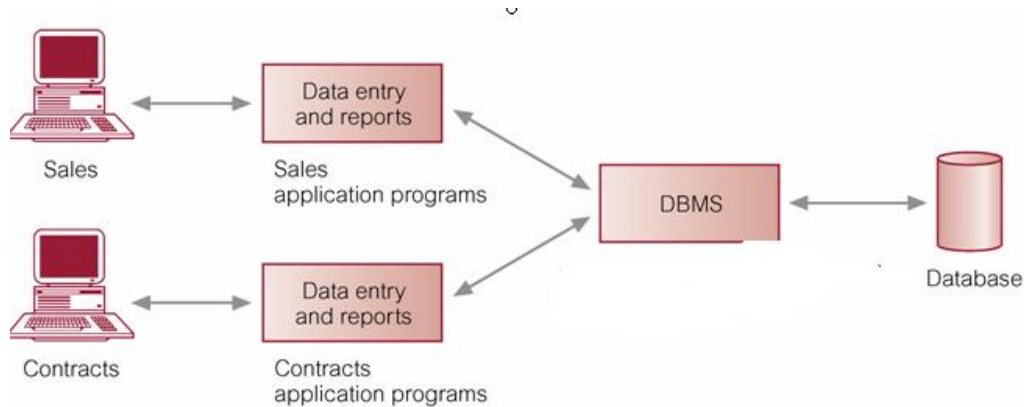
- Main file that encompasses the entire database and that is saved to your hard-drive or floppy disk. (Example: video.mdb )

### Data types:

- Data types are the properties of each field. A field only has 1 data type.
- Most common data types in MS Access are:
  - **Text** Use for text or combinations of text and numbers, such as addresses, or for numbers that do not require calculations, such as phone numbers or postal codes (255 characters)
  - **Memo** Use for lengthy text and numbers, such as notes. Stores up to 63,999 characters
  - **Number** Use for data to be included in mathematical calculations, except money
  - **Date/Time** Use for dates and times
  - **Currency** Use for currency values and to prevent rounding off during calculations.
  - **AutoNumber** Use for unique sequential that are automatically inserted with a new record
  - **Yes/No** Use for data that can be only one of two possible values, such as Yes/No, True/False, On/Off.
  - **OLE Object** Use for OLE objects (such as Microsoft Word documents, Microsoft Excel spreadsheets, pictures, sounds,
  - **Hyperlink** Use for hyperlinks (hyperlink: Colored and underlined text or a graphic that you click to go to a file, a location in a file, a Web page on the World Wide Web, or a Web page on an intranet. Stores up to 2048 characters.
  - **Lookup Wizard** Use to create a field that allows you to choose a value from another table or from a list of values using a combo box

## Database Management System (DBMS)

- A **database management system (DBMS)** refers to the technology for creating and managing databases. **DBMS** is a software tool to organize (create, retrieve, update and manage) data in a database. The main aim of a **DBMS** is to supply a way to store up and retrieve database information that is both convenient and efficient.



- **RDBMS** stands for **Relational Database Management System**. It organizes data into related tables. Tables have rows and column.

## Properties of RDBMS

- An RDBMS is easily accessible. You execute commands in them Structured Query Language (SQL) to manipulate data. SQL is standard language for interacting with a RDBMS.
- An RDBMS provides full data independence
- A relational database is a collection of individual, named objects. The basic unit of data storage in a relational database is called a table.
- An RDBMS enables data sharing between users.
- An RDBMS minimizes the redundancy of data. This means that similar data is not repeated in multiple tables

## Lecture # 33

### Components of a Relational Database Management System (RDBMS)

#### 1. Query Language:

- A query language is an easy-to-use computer language for making queries to a database and for retrieving information from a database.
- There are several different query languages; one of the most popular is *Structured Query Language, or SQL*.

#### 2. Data Dictionary:

- The dictionary or catalog stores information about the database itself
- This is data about data or 'metadata'
- The dictionary holds
  - Descriptions of database objects (tables, users, so on)
  - Information about who is using which data (locks)

#### 3. Utilities:

- The DBMS utilities are programs that enable users to maintain the database.

#### 4. Report Generator:

- The report generator aspect of DBMS software simplifies the process of generating an on-screen or printed-out report.

#### 5. Access Security:

- Access security is a feature that allows database administrators (DBA) to specify different access privileges for **different** users of a DBMS.

#### 6. System Recovery:

- It enables the DBA to recover contents of the database in the event of a hardware or software failure.

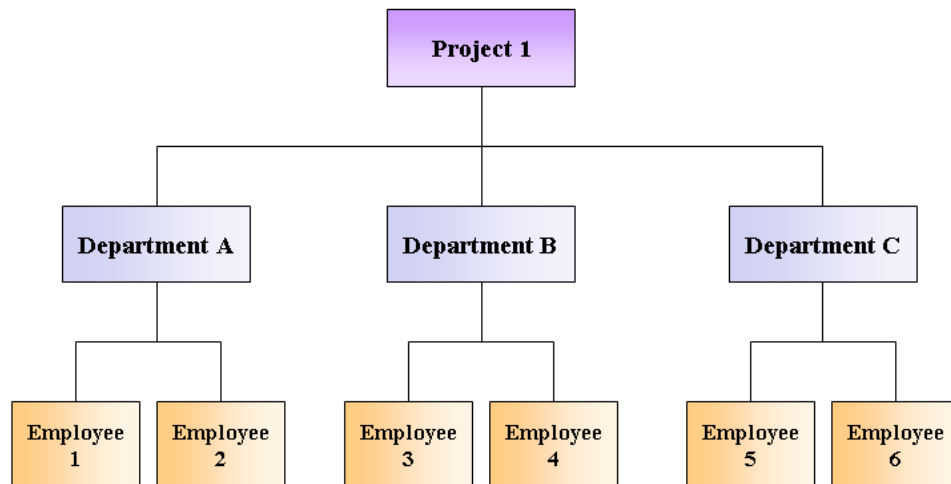
### Database Models

- A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized and manipulated.
- The most popular example of a database model is the relational model, which uses a table-based format.
- Types of database models:

- Hierarchical database model.
- Relational model.
- Network model.
- Object-oriented database model.

### 1) **Hierarchical Database Model**

- A data model in which data are organized in a top-down, or inverted tree structure.

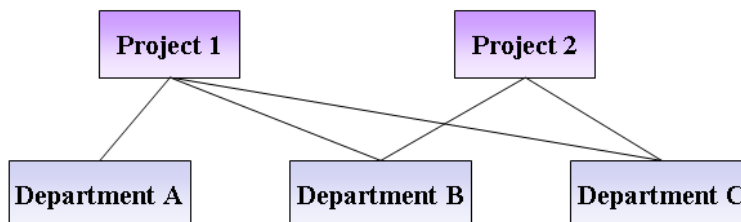


- A hierarchical database consists of the following:

- 1) It contains nodes connected by branches.
- 2) The top node is called the root.
- 3) One parent may have many children

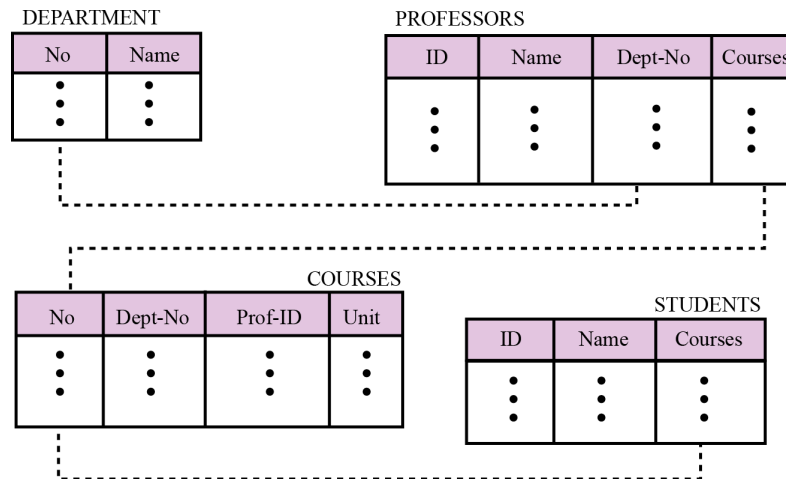
### 2. **Network Database Model**

- In the network model, the entities are organized in a graph, in which some entities can be accessed through several paths



### 3. Relational Database Model

- In the relational model, data is organized in two-dimensional tables called relations. The tables or relations are, however, related to each other.
- An example of the relational model representing a university



### 4. Object-Oriented DBMS:

- These systems can handle objects such as videos, images, pictures, and so on.

## Lecture # 34

### Keys

- A DBMS key is an attribute or set of an attribute which helps you to identify a row in a table.
- They allow you to find the relation between two tables.
- Five common types of Keys are:
  - i. Super Key
  - ii. Primary Key
  - iii. Candidate Key
  - iv. Foreign Key
  - v. Composite Key

**Super key:** A super key is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

**Primary Key:** A primary key helps us to uniquely identify every row in the table.

#### **Rules for defining Primary key:**

- Two rows can't have the same primary key value
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- **Example:** Student's roll number is a Primary Key.

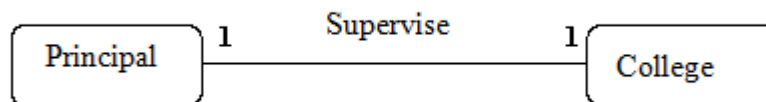
**Candidate Key:** A super key with no repeated attribute is called candidate key.

**Foreign Key:** A foreign key is a column which is added to create a relationship with another table. Every relationship in the model needs to be supported by a foreign key.

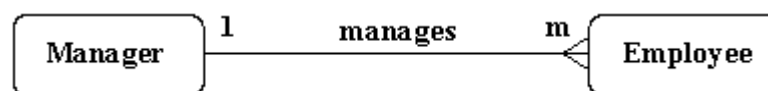
**Composite key:** A key which has multiple attributes to uniquely identify rows in a table is called a composite key.

## Relationships

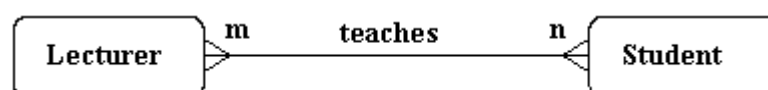
- A Relationship represents an association between two or more tables / entities.
- An example of a relationship would be:
  - Employees are assigned to projects
  - Projects have subtasks
  - Departments manage one or more projects
- The basic types of relationships are:
  - **One-to-One**
    - Each entity in the relationship will have exactly one related entity
    - A principal can only supervise one college, and a college have only one principal, so it is a one to one (1:1) relationship



- **One-to-Many**
  - An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity
  - one manager manages many employees, but each employee only has one manager, so it is a one to many (1:m) relationship



- **Many-to-Many**
  - Entities on both sides of the relationship can have many related entities on the other side
  - One lecturer teaches many students and a student is taught by many lecturers, so it is a many to many (m:n) relationship



## Lecture # 35

### Database Administrator (DBA)

- The database Administrator (DBA) is the person who makes the strategic and policy decisions regarding the data of the enterprise.
- The DBA is responsible for the design, operation and management of the database.
- **Database administrator's duties / Responsibilities:**
  - Storage structure and access method definition
  - Granting users authority to access the database
  - Backing up data
  - Monitoring performance and responding to changes
  - Database tuning

### DBMS Languages

- SQL (Structured Query Language) is a relational DB language includes a combination of DDL(Data Definition Language), DCL (Data Control Language) and DML (Data Manipulation Language).
- **DDL (data definition language)**
  - It permits specification of data types, structures and any data constraints.
- **DML (data manipulation language)**
  - It is used in retrieving, inserting, deleting and manipulating data.
- **DCL (data control language)**
  - It is used to grants rights to users.



## **Important Questions:**

### **Q 1. Define relation.**

**Ans.** In relational database, the table in which data is stored is called a relation. Collection of rows and column is called table. Each intersection of a row and column is called cell. Table contains the descriptive information about an entity. Table is also called relation. Each file in a file management system corresponds to a table in database management system.

### **Q 2. What is an Entity?**

**Ans.** Anything about which we want to store data is called entity. It can be a person, place or event etc. Entity always has a unique name within a domain.

### **Q 3. What is a key?**

**Ans.** A key field is a field or set of fields of a database table which together form a unique identifier for a database record. The aggregate of these fields is usually referred to simply as "the key". A key field also defines searches.

### **Q 4. Define primary key.**

**Ans.** In a relation, the attribute or a combination of attributes that uniquely identifies a row or a record. e.g. A CNIC No., ISBN, student roll number, etc.

### **Q 5. Define secondary key.**

**Ans.** A secondary key is a non-unique field that is used as a secondary or alternate key. Some times records are required to be accessed by a field other than the primary key. In these situations another key that is used is called secondary key or alternate key.

### **Q 6. Define candidate key.**

**Ans.** There can be more than one keys or key combinations that qualify to be selected as primary key. In a relation there can be only one primary key at a time. Rest of the keys or key combinations are called candidate keys.

### **Q 7. Define composite key.**

**Ans.** Composite key consists of two or more than two fields. Composite key is also designated as a primary key. It is created in a situation when no single field fulfills the property of uniqueness. To make unique more than one field are combined and used as primary key.

**Q 8. Who is end user?**

**Ans.** It is the person who uses the database management system for his need. He must have knowledge of information technology. He does not need to have the detail knowledge of the computer system. He should be aware of the usage details of the software he intends to use.

**Q 9. Who is database administrator?**

**Ans.** A database administrator (DBA) is a person who is responsible for the environmental aspects of a database. In general, these include:

- **Recoverability:** Creating and testing backups.
- **Integrity:** Verifying or helping to verify data integrity.
- **Security:** Defining and/or implementing access controls to the data.
- **Availability:** Ensuring maximum up time.
- **Performance:** Ensuring maximum performance given budgetary constraints.
- **Development and testing support:** Helping programmers and engineers to efficiently utilize the database.

**Q 10. List two properties of a relation.**

**Ans.** Properties of a relation are:

- It has unique column names.
- The order of column is insignificant.
- The order of row is insignificant.

**Q 11. Discuss the data manipulation in DBMS system?**

**Ans.** Data manipulation of database management system is different from file management system. In database management system:

- Data is stored in relation or tables.
- A database may have more than one relation with unique names.
- Relations in a database relate to each other using primary and foreign keys.
- DBMS uses index to quickly access the data stored in relation.
- Database query language i.e. SQL is used for data manipulation in database.

## **MULTIPLE CHOICE QUESTIONS**

1. The \_\_\_\_\_ loop executes at least once.  
a) For                                      b) While  
c) Do-While                                d) None of above
  
2. MS-Access is \_\_\_\_\_ type of database management system.  
a) Network                                b) Hierarchical  
c) Complex                                d) Relational
  
3. The multiply operator (\*) is \_\_\_\_\_ type of operator.  
a) Arithmetic                              b) Relational  
c) Assignment                             d) Logical
  
4. The \_\_\_\_\_ loop means loop inside loop.  
a) Recursive                                b) Nested  
c) Simple                                    d) Complex
  
5. \_\_\_\_\_ key is used to uniquely identify a record.  
a) Foreign                                  b) Primary  
c) Composite                                d) Native
  
6. In C language \_\_\_\_\_ function is used to clear the screen.  
a) ceil( )                                    b) floor( )  
c) abs( )                                    d) clrscr( )
  
7. To use printf and scanf functions \_\_\_\_\_ header file must be included.  
a) stdio.h                                  b) math.h  
c) string.h                                 d) alloc.h
  
8. DBMS stands for \_\_\_\_\_.  
a) Database Management System    b) Database Monitoring System  
c) Database Migration System    d) Data Bounded Monitoring System
  
9. In C language \_\_\_\_\_ format specifier is used for integer number.  
a) %f                                        b) %Lf  
c) %c                                        d) %d

10. The function which calls itself is called \_\_\_\_\_ function.  
a) Recursive                      b) Nested  
c) Simple                         d) Complex
11. SQL stands for \_\_\_\_\_.  
a) Standard Query Language      b) Structured Query Language  
c) Standard Question Language    d) Standard Quantum Language
12. In C language \_\_\_\_\_ function is used to compute the Cosine of a value.  
a) floor( )                      b) sin( )  
c) cos( )                         d) ceil( )
13. C language belongs to \_\_\_\_\_ generation of languages.  
a) First                          b) Fifth  
c) Sixth                         d) Third
14. The remainder operator (%) is \_\_\_\_\_ type of operator.  
a) Arithmetic                      b) Relational  
c) Assignment                      d) Logical
15. In C language \_\_\_\_\_ function is used to return absolute value.  
a) clrscr( )                      b) getch( )  
c) abs( )                         d) exp( )
16. In flowchart \_\_\_\_\_ symbol represents start or finish of program.  
a) Diamond                      b) Oval  
c) Rectangle                      d) Square
17. The collection of similar (homogeneous) data items is called \_\_\_\_\_.  
a) Structure                      b) Array  
c) Variable                         d) Pointer
18. String functions are accessible through the \_\_\_\_\_ header file.  
a) stdio.h                      b) math.h  
c) string.h                         d) alloc.h

19. The \_\_\_\_\_ statement is used to exit the switch structure.
- a) goto                                      b) default
  - c) return                                    d) break
20. Parallelogram symbol in a flowchart represents \_\_\_\_\_.
- a) Decision                                b) Process
  - c) Start/stop                              d) Input/output
21. A global variable is defined in \_\_\_\_\_.
- a) main() function                        b) In second function
  - c) In any function                        d) Outside of any function
22. Which of the following is an arithmetic operator
- a) +    b) &
  - c) &&                                        d) <
23. A relational operator is used to
- a) Combine values                        b) Compare values
  - c) Delete values                          d) Interchange the values
24. Software that translates assembly language into machine language is called
- a) Loader                                    b) Linker
  - c) Debugger                                d) Assembler
25. The collection of records is called:
- a) Query                                    b) Field
  - c) Index                                    d) File
26. The language computer can understand is only binary code made up of 1s and 0s, this language is
- a) C    b) Assembly
  - c) Machine                                d) Fortran
27. C is an example of:
- a) Compiler                                b) Interpreter
  - c) Antivirus                                d) Spreadsheet

28. Software that translates the entire program into machine language before its execution is:

- a) Word processor                      b) Database
- c) Compiler                              d) Spreadsheet

29. Size of a float variable is \_\_\_\_\_ bytes.

- a) Three                                      b) Twelve
- c) Sixteen                                   d) Four

30. Range of int type variables is from \_\_\_\_\_ to \_\_\_\_\_.

- a) -32768 to 32767                      b) -1024 to 1024
- c) -65536 to 65537                      d) 0 to 2048

31. Which of the following is not a relational operator:

- a) &&    b) >=
- c) <=    d) >

32. Which of the following is/are not correct declarations:

- a) double d;                                  b) integer i;
- c) float f;                                    d) int i;

33. Which of the following is not correctly initialized

- a) float y = 2.3;                              b) int z = 2;
- c) char c = 'pakistan';                      d) double y = 8;

34. The expression in which relational operators are used is called:

- a) Arithmetic                                  b) Erroneous
- c) Relational                                   d) Logical

35. A \_\_\_\_\_ error can be caused by a simple typographical error.

- a) Syntax                                      b) Compilation
- c) Run-time                                    d) Logical

36. There are two types of languages high level languages and \_\_\_\_\_ level languages.

- a) Very high                                   b) Complex
- c) Low    d) Periodic

37. Which Access object is considered primary?
- a) Table                      b) Query
  - c) Report                     d) Form
38. Compiler and interpreters are types of \_\_\_\_\_.
- a) Word Processors              b) Language translators
  - c) Antivirus                      d) Spreadsheets
39. && and || are termed as \_\_\_\_\_.
- a) Arithmetic operators              b) Relational operators
  - c) Assignment operators              d) Logical operators
40. The rectangle symbol in a flowchart represents:
- a) Beginning of program              b) Decision
  - c) Connection                      d) Process
41. Strings are stored as array of:
- a) Characters                      b) Numbers
  - c) Logical values                      d) Expressions
42. A variable that is known only within the function in which it is defined is called:
- a) Global variable                      b) Public variable
  - c) Pure variable                      d) Local variable
43. Loop is also called:
- a) Division                      b) Modification
  - c) Termination                      d) Iteration
44. An association established between common fields in two tables is called:
- a) Bond                      b) Relationship
  - c) Friendship                      d) Union
45. In MS-Access \_\_\_\_\_ is a way to present data in a printed format.
- a) Query                      b) Table
  - c) Form                      d) Report

46. A \_\_\_\_\_ is a type of database which stores information in tables.
- a) Hierarchical database                      b) Network database
  - c) Relational database                        d) Tree database
47. A value that indicates unknown or missing data in a field is called:
- a) Null value                                      b) Absolute value
  - c) Composite value                              d) Logical value
48. The data item that operators act upon is called \_\_\_\_\_.
- a) Mod    b) Argument
  - c) Parameter                                      d) Operand
49. The statement that is used to bypass the remainder of the loop and go back to beginning of loop:
- a) break    b) continue
  - c) exit    d) return
50. An escape sequence always begins with a \_\_\_\_\_ sign.
- a) &    b) \
  - c) /    d) \*
51. Type int variables comprise of \_\_\_\_\_ number of bytes.
- a) One byte                                        b) Two bytes
  - c) Four bytes                                      d) Eight bytes
52. Every C language statement must end with \_\_\_\_\_ sign.
- a) /    b) \*
  - c) }    d) ;
53. The #include is used in the beginning of a C language program to include \_\_\_\_\_ file.
- a) Header                                        b) Object
  - c) Source                                        d) Database
54. C language was developed by the person named \_\_\_\_\_.
- a) Charles Babbage                              b) Dennis Ritchie
  - c) Bill Gates                                      d) Robert Lafore



55. To hold alphabetic type values \_\_\_\_\_ type of variable is used.
- a) int                                      b) float
  - c) double                                  d) char
56. IDE stands for \_\_\_\_\_.
- a) Internet Development Elite                      b) Integrated Development Environment
  - c) Internal Development Enterprises              d) Integrated Display Enforcement
57. To run a C language program in IDE \_\_\_\_\_ keys are used.
- a) CTRL-F1                                  b) ALT-F1
  - c) CTRL-F9                                  d) F2
58. C language was developed in the year \_\_\_\_\_.
- a) 1984                                      b) 1972
  - c) 1922                                      d) 1914
59. In flowchart \_\_\_\_\_ symbol is used to define a decision box.
- a) Diamond                                  b) Oval
  - c) Rectangle                                  d) Square
60. The \_\_\_\_\_ loop checks the condition at the end of loop.
- a) For    b) While
  - c) Do-While                                  d) None of above