Biyani's Think Tank

Concept based notes

Computer Programming & Information Technology

B.Tech.First Semester

Pukhraj Deep Chouhan
Asst. Prof.
Biyani International Institute of Engg. & Tech.
Jaipur

Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph. : 0141-2338371, 2338591-95 ı Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website : www.biyanithinktank.com; www.biyanicolleges.org

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concept of the topic. The book is self-explanatory and adopts the "Teach Yourself" style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the reader for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chiarman*,Dr. Sanjay Biyani, *Director (Acad.)* Biyani Group of Colleges & Mr. Manish Biyani, Director Biyani International Institute of Engineering & Technology, who is the backbone and main concept provider and also have been constant source of motivation throughout this endeavour. We also extend our thanks to M/s. Hastlipi, Omprakash Agarwal/Sunil Kumar Jain, Jaipur, who played an active role in co-ordinating the various stages of this endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and the students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

Content

Syllabus

B.Tech., First Semester

Computer programming & Information Technology

Unit – I

Introduction: Stored Program Architecture of Computers, Evolution of Processors (In terms of word length & Speed only), Storage Device- Primary Memory and Secondary Storage, Working Principle of Primary Storage devices- RAM, ROM, PROM, EPROM, EEPROM, Random, Direct, Sequential access methods.

Language Translators – Concept of High-Level, Assembly and Low Level programming languages. Working of Assembler, Interpreter and compiler. Representing Algorithms through flow chart , pseudo code, step by step etc.

Unit – II

Number System: Data Representation, Concept of radix and representation of numbers in radix r with special cases of r=2, 8, 10 and 16 with conversion from radix r1 to radix r2. r's and (r-1)'s complement. Representation of Integer in sign-magnitude, signed 1's and 2's complement. Floating point representation . Concept of bias and normalization . Representation of alphabets .

Binary Codes: Binary arithmetic, Addition and subtraction of Integers and floating point numbers. Multiplication of Integers . Gray code, BCD 8421 and 2421, Excess-3 and Excess-3 gray codes.

Unit – III

Programming in C: Structure of C Program, Concept of Preprocessor, Macro Substitution, Intermediate code, Object Code, Executable Code. Compilation Process,

Basic Data types, Importance of braces ({ }) in C Program, enumerated data type, Identifiers, Scope of Variable, Storage Class, Constants, Expressions in C, Type Casting, Control Statements, printf( ), scanf ( ), reading single character. Command Line Arguments.

Unit – IV

Arrays in C, Pointers, Using pointers to represent arrays, Dynamic Memory allcation,

Structures, using typedef, Arrays of Structures & pointers, File Handling (Opening in

different modes & closing of file, fscanf & fprintf only).

Unit – V

Functions in C, Passing Parameters (By value & Reference), using returned data, Passing arrays, structures, array of structures, pointer to structures etc., passing characters and strings, The void pointer.

Unit I
Introduction & Language Translators

Q.1: What is Computer?

Ans: Computer is an electronic device. As mentioned in the introduction it can do arithmetic calculations faster. But as you will see later it does much more than that. It can be compared to a magic box, which serves different purpose to different people. For a common man computer is simply a calculator, which works automatic and quite fast. For a person who knows much about it, computer is a machine capable of solving problems and manipulating data. It accepts data, processes the data by doing some mathematical and logical operations and gives us the desired output.

Therefore, we may define *computer as a device that transforms data*. Data can be anything like marks obtained by you in various subjects. It can also be name, age, sex, weight, height, etc. of all the students in your class or income, savings, investments, etc., of a country. Computer can be defined in terms of its functions. It can i) accept data ii) store data, iii) process data as desired, and iv) retrieve the stored data as and when required and v) print the result in desired format.

Q.2: What are characteristic of computer?
Ans:

1. Speed: - As you know computer can work very fast. It takes only few seconds for calculations that we take hours to complete. You will be surprised to know that computer can perform millions (1,000,000) of instructions and even more per second. Therefore, we determine the speed of computer in terms of microsecond (10-6 part of a second) or nanosecond (10 to the power -9 part of a second). From this you can imagine how fast your computer performs work.

2. Accuracy: - The degree of accuracy of computer is very high and every calculation is performed with the same accuracy. The accuracy level is

determined on the basis of design of computer. The errors in computer are due to human and inaccurate data.

3. Diligence: - A computer is free from tiredness, lack of concentration, fatigue, etc. It can work for hours without creating any error. If millions of calculations are to be performed, a computer will perform every calculation with the same accuracy. Due to this capability it overpowers human being in routine type of work.

4. Versatility: - It means the capacity to perform completely different type of work. You may use your computer to prepare payroll slips. Next moment you may use it for inventory management or to prepare electric bills.

5. Power of Remembering: - Computer has the power of storing any amount of information or data. Any information can be stored and recalled as long as you require it, for any numbers of years. It depends entirely upon you how much data you want to store in a computer and when to lose or retrieve these data.

6. No IQ: - Computer is a dumb machine and it cannot do any work without instruction from the user. It performs the instructions at tremendous speed and with accuracy. It is you to decide what you want to do and in what sequence. So a computer cannot take its own decision as you can.

7. No Feeling: - It does not have feelings or emotion, taste, knowledge and experience. Thus it does not get tired even after long hours of work. It does not distinguish between users.

8. Storage: - The Computer has an in-built memory where it can store a large amount of data. You can also store data in secondary storage devices such as floppies, which can be kept outside your computer and can be carried to other computers.

Q.3: What are Basic Computer Operations?

Ans: - A computer as shown in Fig. 2.1 performs basically five major operations or functions irrespective of their size and make. These are 1) it accepts data or instructions by way of input, 2) it stores data, 3) it can process data as required by the user, 4) it gives results in the form of output, and 5) it controls all operations inside a computer. We discuss below each of these operations.

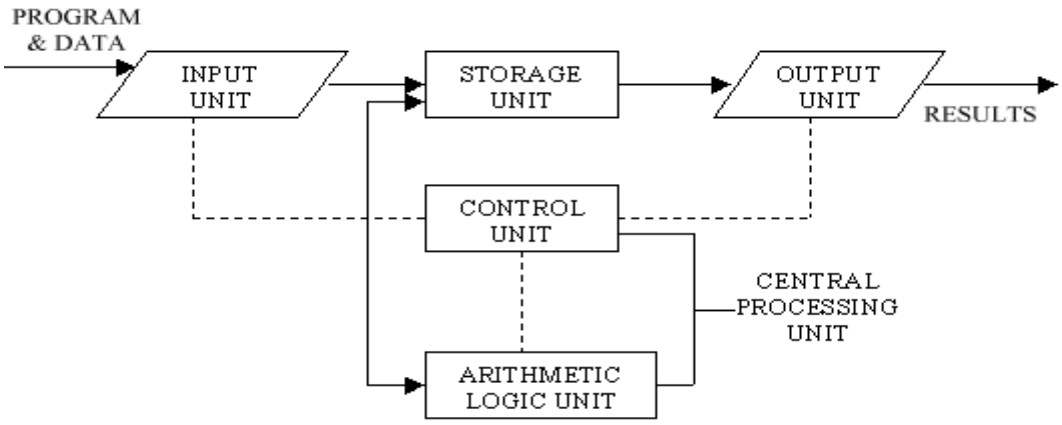


Fig : ***Basic computer Operations***

1. Input: This is the process of entering data and programs in to the computer system. You should know that computer is an electronic machine like any other machine which takes as inputs raw data and performs some processing giving out processed data. Therefore, the input unit takes data from us to the computer in an organized manner for processing.

2. Storage: The process of saving data and instructions permanently is known as storage. Data has to be fed into the system before the actual processing starts. It is because the processing speed of Central Processing Unit (CPU) is so fast that the data has to be provided to CPU with the same speed. Therefore the data is first stored in the storage unit for faster access and processing. This storage unit or the primary storage of the computer system is designed to do the above functionality. It provides space for storing data and instructions.

The storage unit performs the following major functions:

- All data and instructions are stored here before and after processing.
- Intermediate results of processing are also stored here.

3. Processing: The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit.

4. Output: This is the process of producing results from the data for getting useful information. Similarly the output produced by the computer after processing must also be kept somewhere inside the computer before being given to you in human readable form. Again the output is also stored inside the computer for further processing.

5. Control: The manner how instructions are executed and the above operations are performed. Controlling of all operations like input, processing and output are performed by control unit. It takes care of step by step processing of all operations inside the computer.

Q:4 What is Stored Program Architecture?
Ans:

The defining feature of modern computers which distinguishes them from all other machines is that they can be programmed. That is to say that a list of instructions (the program) can be given to the computer and it will store them and carry them out at some time in the future.

In most cases, computer instructions are simple: add one number to another, move some data from one location to another, send a message to some external device, etc. These instructions are read from the computer's memory and are generally carried out (executed) in the order they were given. However, there are usually specialized instructions to tell the computer to jump ahead or backwards to some other place in the program and to carry on executing from there. These are called "jump" instructions (or branches). Furthermore, jump instructions may be made to happen conditionally so that different sequences of instructions may be used depending on the result of some previous calculation or some external event. Many computers directly support subroutines by providing a type of jump that "remembers" the location it jumped from and another instruction to return to the instruction following that jump instruction.

Program execution might be likened to reading a book. While a person will normally read each word and line in sequence, they may at times jump back to an earlier place in the text or skip sections that are not of interest. Similarly, a computer may sometimes go back and repeat the instructions in some section of the program over and over again until some internal condition is met. This is called the flow of control within the program and it is what allows the computer to perform tasks repeatedly without human intervention.

Comparatively, a person using a pocket calculator can perform a basic arithmetic operation such as adding two numbers with just a few button presses. But to add together all of the numbers from 1 to 1,000 would take thousands of button presses and a lot of time — with a near certainty of making a mistake. On the other hand, a computer may be programmed to do this with just a few simple instructions.

Once told to run this program, the computer will perform the repetitive addition task without further human intervention. It will almost never make a mistake and a modern PC can complete the task in about a millionth of a second.

However, computers cannot "think" for themselves in the sense that they only solve problems in exactly the way they are programmed to. An intelligent human faced with the above addition task might soon realize that instead of actually adding up all the numbers one can simply use the equation

1+2+3+…+n = {{n(n+1)} \over 2}

and arrive at the correct answer (500,500) with little work. In other words, a computer programmed to add up the numbers one by one as in the example above would do exactly that without regard to efficiency or alternative solutions.

Q.5: What are Storage Devices?

Ans: Computer data storage, often called storage or memory, refers to computer components and recording media that retain digital data used for computing for some interval of time. Computer data storage provides one of the core functions of the modern computer, that of information retention. It is one of the fundamental components of all modern computers, and coupled with a central processing unit (CPU, a processor).

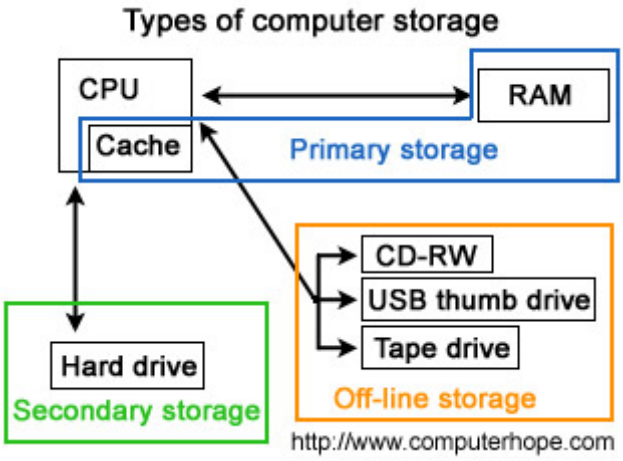


Fig :

Q.6: What are different types of storage?

Ans: Two type of basic storage we have in computer are primary and secondary storage:-

Primary Storage: - Primary storage is a storage location that holds memory for short periods of times while the computer is on. For example, computer RAM (random-access memory) and cache are both examples of a primary storage device. This type of storage is the fastest type of memory in your computer and is used to store data while it's being used. For example, when you open a program data is moved from the secondary storage into the primary storage. It is also known as internal memory and main memory.

Secondary Storage: - Secondary storage is a storage medium that holds information until it is deleted or overwritten regardless if the computer has power. For example, a floppy disk drive and hard disk drive are both good examples of secondary storage devices. As can be seen by the below picture there are three different types of storage on a computer, although primary storage is accessed much faster than secondary storage because of the price and size limitations secondary storage is used with today's computers to store all your programs and your personal data. It is also known as external memory and auxiliary storage. Off-line storage in Fig 1.2 could be considered secondary storage, we've

separated these into their own category because these types of media can be easily removed from the computer and stored elsewhere.

Q.7: Explain few primary storage devices?

Ans: - Generally used primary storage devices are:-

1. Random Access Memory (RAM): The primary storage is referred to as random access memory (RAM) because it is possible to randomly select and use any location of the memory directly store and retrieve data. It takes same time to any address of the memory as the first address. It is also called read/write memory. The storage of data and instructions inside the primary storage is temporary. It disappears from RAM as soon as the power to the computer is switched off. The memories, which lose their content on failure of power supply, are known as **volatile** memories .So now we can say that RAM is volatile memory.

2. Read Only Memory (ROM): There is another memory in computer, which is called Read Only Memory (ROM). Again it is the Ics inside the PC that form the ROM. The storage of program and data in the ROM is permanent. The ROM stores some standard processing programs supplied by the manufacturers to operate the personal computer. The ROM can only be read by the CPU but it cannot be changed. The basic input/output program is stored in the ROM that examines and initializes various equipment attached to the PC when the switch is made ON. The memories, which do not lose their content on failure of power supply, are known as **non-volatile** memories. ROM is non-volatile memory.

3. PROM There is another type of primary memory in computer, which is called Programmable Read Only Memory (PROM). You know that it is not possible to modify or erase programs stored in ROM, but it is possible for you to store your program in PROM chip. Once the program are written it cannot be changed and remain intact even if power is switched off. Therefore programs or instructions written in PROM or ROM cannot be erased or changed.

4. EPROM: This stands for Erasable Programmable Read Only Memory, which overcome the problem of PROM & ROM. EPROM chip can be programmed time and again by erasing the information stored earlier in it. Information stored in EPROM exposing the chip for some time ultraviolet light and it erases chip is reprogrammed using a special programming facility. When the EPROM is in use information can only be read.

5. EPROM: - This stand for electrically erasable programmable read-only memory. EEPROM is a special type of PROM that can be erased by exposing it to an electrical charge. Like other types of PROM, EEPROM retains its contents even when the power is turned off. EPROM is similar to flash

memory (sometimes called flash *EEPROM*). The principal difference is that EEPROM requires data to be written or erased one byte at a time whereas flash memory allows data to be written or erased in blocks. This makes flash memory faster

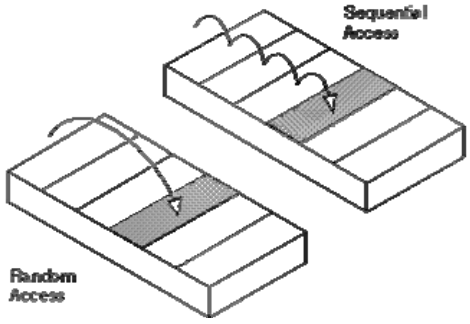
6. Cache Memory: The speed of CPU is extremely high compared to the access time of main memory. Therefore the performance of CPU decreases due to the slow speed of main memory. To decrease the mismatch in operating speed, a small memory chip is attached between CPU and Main memory whose access time is very close to the processing speed of CPU. It is called CACHE memory. CACHE memories are accessed much faster than conventional RAM. It is used to store programs or data currently being executed or temporary data frequently used by the CPU. So each memory makes main memory to be faster and larger than it really is. It is also very expensive to have bigger size of cache memory and its size is normally kept small.

7. **Registers:** The CPU processes data and instructions with high speed, there is also movement of data between various units of computer. It is necessary to transfer the processed data with high speed. So the computer uses a number of special memory units called *registers.* They are not part of the main memory but they store data or information temporarily and pass it on as directed by the control unit.

Q.8: What is Access Method? Explain Different type of Access Methods?

Ans: - In computing, an access method is a program or a hardware mechanism that moves data between the computer and an outlying device such as a hard disk (or other form of storage) or a display terminal. The term is sometimes used to refer to the mechanics of placing or locating specific data at a particular place on a storage medium and then writing the data or reading it. It is also used to describe the way that data is located within a larger unit of data such as a data set or file.

There are two type of access method *random access and sequential access*.



To go from point A to point Z in a sequential-access system, you must pass through all intervening points. In a random-access system, you can jump directly to point Z.Disks are random access media, whereas tapes are sequential access media.

The terms *random access* and *sequential access* are often used to describe data files. A random-access data file enables you to reador write information anywhere in the file. In a sequential-access file, you can only read and write information sequentially, starting from the beginning of the file.

Both types of files have advantages and disadvantages. If you are always accessing information in the same order, a sequential-access file is faster. If you tend to access information randomly, random access is better.

Unit II
Programming in C

Q.1: Define C programming language? Why we use C programming language?

Ans.: C is a general-purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system.

Although C was designed for implementing system software, it is also widely used for developing portable application software.

C is one of the most popular programming languages of all time and there are very few computer architectures for which sa C compiler does not exist. C has greatly influenced many other popular programming languages, most notably C++, which began as an extension to C.

C is an imperative (procedural) systems implementation language. It was designed to be compiled using a relatively straightforward compiler, to provide low-level access to memory, to provide language constructs that map efficiently to machine instructions, and to require minimal run-time support. C was therefore useful for many applications that had formerly been coded in assembly language.

Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with few changes to its source code. The language has become available on a very wide range of platforms, from embedded microcontrollers to supercomputers.

C's design is tied to its intended use as a portable systems implementation language. It provides simple, direct access to any addressable object (for example, memory-mapped device control registers), and its source-code expressions can be translated in a straightforward manner to primitive machine operations in the executable code.
Characteristics:-
Like most imperative languages in the ALGOL tradition, C has facilities for structured programming and allows lexical variable scope and recursion, while a static type system prevents many unintended operations. In C, all executable code is contained within functions. Function parameters are always passed by value. Pass-by-reference is simulated in C by explicitly passing pointer values. Heterogeneous aggregate data types (struct) allow related data elements to be combined and manipulated as a unit. C program source text is free-format, using the semicolon as a statement terminator.
C also exhibits the following more specific characteristics:
Variables may be hidden in nested blocks Partially weak typing. for instance, characters can be used as integers
Low-level access to computer memory by converting machine addresses to typed pointers Function and data pointers supporting ad hoc run-time polymorphism array indexing as a secondary notion, defined in terms of pointer arithmetic
A preprocessor for macro definition, source code file inclusion, and conditional compilation Complex functionality such as I/O, string manipulation, and mathematical functions consistently delegated to library routines A relatively small set of reserved keywords A large number of compound operators, such as +=, -=, *=, ++ etc.

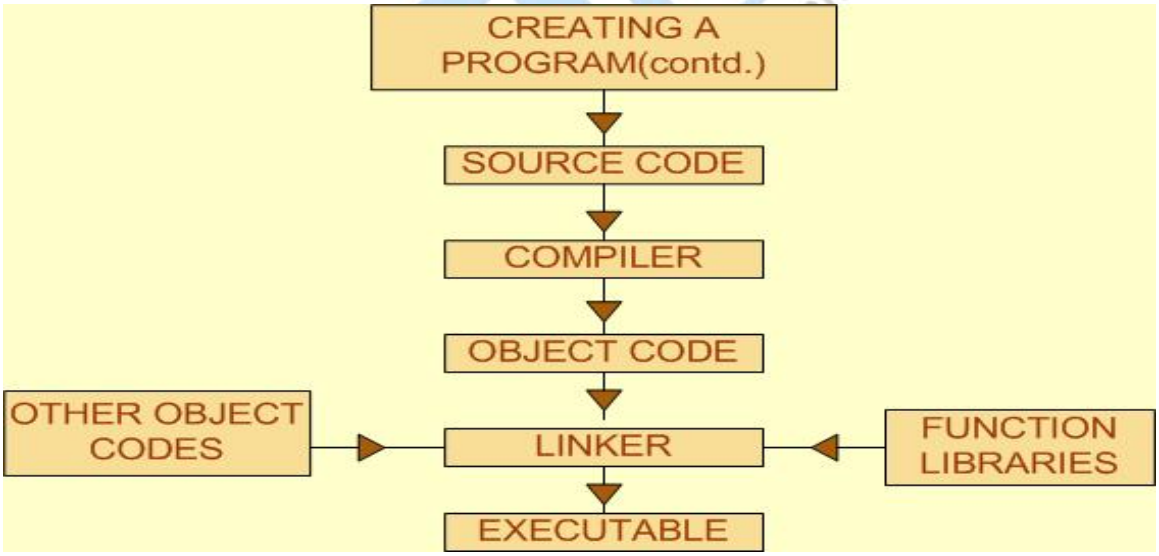
Q.2: Define the procedure to create a program in C programming language?

Ans: There are many "languages" like, for example C, Fortran, PASCAL etc., that help us to convert an algorithm in to something that a computer can understand. We will focus here only on C programming
So. let us start looking at how do we create a "C program".There are basically three steps involved in converting your ideas into what is to be done to make it a working program.

There are three steps involved in converting your idea of what is to be done to a working program

1. **Creating a source code in the form of the text file acceptable to the compiler.**
2. **Invoking the compiler to process the source code and produce an object file.**
3. **Linking all the object files and libraries to produce an executable**

We will look at these parts in some more detail here.
The source code is where you express the the whole computation you want to do into a series of instructions.
The tradition is to use the C source code with an extension .c For example myfile.c . Some compilers insist on this extension. This is an acsii file that you can create using any one of the editors that you are familiar with. We will look at the detailed structure of this code in a little while .
We may have different parts of the code in different files. We then compile them separately or together to form object files. During this operation the compiler checks and convert your source code to a language understandable to the particular processor that is used in the computer. These object files that comes with an extension .o, for example myfile.o , are in binary format.
The next step is to link these object files and other libraries and header files etc., to produce an executable. This executable file contains a set of instruction in a language that the processor can understand and is the one which we run on the computer. We will see the use of header files and system libraries in the examples given below.
Before proceed let us summarize the steps that takes us to run a program in the form of a flow chart.



As we said earlier it is the tradition to write a C program file name ending with the extension .C.

Example: Below we give a program that computes the square root of a number and prints it on the scree. Since it is a C program we name it as "myfile.c" , that is with an extension ".c".

```
1   #include <math.h>

2   #include <stdio.h>
3   main()
  {
4     float x,y;

5         scanf("%f\n",&x);

6           y=sqrt(x);

7         printf("%f\n",y);

    }
```

Before we proceed to the compilation of this program let us take a minute to see what does each of the lines mean?

Lines 1 and 2 are header files. These files contain the math library functions and the input output command functions. We need the math function library to use functions like "sqrt", "log", "power" etc. We need the stdio library to use functions like "scanf" and "printf" which reads and prints the data from the screen. More on this a little later.

The program given above just has a main body. Line 3 is declares the starting on this part. The lines inside the { } following this are the part of the main program.

Line 4 is the declaration of the variables. Again we will see more on the variable types later in this course. In this particular example we declare variables "x,y" as floating points.

Line 5 reads the value of the variable from the screen.

Line 6 computes the square root of the variable "x" and put the value into variable "y".

Line 7 prints out this value of y.

We first compile this code to produce an object code using the command cc -c myfile.c or an equivalent command available at your machine.

The object file produced by this, myfile.o is then linked with the system math library using the command, cc -lm myfile.o -o myfile.exe , to produce the executable myfile.exe.

(the second term in the command above "-lm" links the object file to the math library. The file name that comes just after -o is the name of the executable).
We can combine compile and link into one step as will see in later part of the course.
Example: Let us now look at a C program that uses the library and another function.

```
#include <math.h>
#include <stdio.h>
main()
{
  float x,y;
    ("%f\n",&x);
          y=sqrt(x);
          printf("%f\n",y);
          printit();
 printf()
{
      printf("The program is Over");
}
```

This simple C code adds an additional component to the one we saw earlier, it calls a function "printit". As before you can see the "include" files, or what are called header files. Then we will see the main part of the program. This program is basically computing square root of the variable x and it is printing it out on the screen. Then it is calling some function, which prints out a statement.Here we see on of the main feature of a C program. The main part does some of the calculations and it calls another function. The function we called, named printit, which simply prints the sentence "the program is over." can be put it a separate file. Let is call it "print.c". We will see that how it can be compiled and how it can be executed.There are two ways to compile this program.To compile these programs we could type **cc -c sample.c and cc -c print.c** separately.This commands will create sample.o and print.o files. We can create the executable by typing the command.**cc sample.o print.o -lm -o a.out** In this linking command we have put together the two files and also included the math library file m.a We can now execute the program by typing a.out.You could try variations of these commands and programs to figure out why we need the math.h file and why we need to include -lm in the linker etc. Also try it for different math functions, for example calculating the logarithm or trigonometric functions etc.Another way to compile and generate an executable from this is to simply type cc sample.c print.c -lm -o a.out In this the compilation and linking is put together.Let us try another program to illustrate the use of math library and the compilation and linking procedure

Q.3: Define the anatomy of C program ?

Ans: The anatomy of a program

Let us look at the anatomy of a program once again. It can be summarized as below.Compiler preprocessor commands :-

```
main function name()
{
    variable declaration
     code
}
sub-function-name (arguments)
argument declarations
{
    variable declarations
    code
}
```

We have some compiler preprocessor commands. This includes various #include files. Then comes the main function. Some name can also be given to the main function. Then, we have the variable declarations used in the main code. Then we have sub-functions.

In the previous example, we had the sub functions as a separate files. But you can also include them it into the main code. In this case, in one file, we can have main program of the main function and sub-functions or separately. It is a good idea to have the sub-functions separately. We also compile them n link them separately. It is always safer to do that. For large programs, it is good to have sub functions separately. Sub function has also the same structure like main function. It might have some arguments which would be passed into it. Then comes argument declaration. Then we have some variable declarations which is only private to sub functions only. We will see the examples of these.

Before going to the example, let us look at each one these in detail.

The standard C compiler has four filters. The first one is known as the preprocessor. This part handles the inclusion of files mentioned using the #include command, definition of constants, macro definition etc.

The second filter checks the language statements, generates a symbol table and reports any errors found.

The third part generates the code and a fourth filter optimizes the code for better performance
The fourth one optimizes the function. It improves the code and modifies the code and so that it runs faster. This is an optional part of the compiler.

Q.4: Define basic data types those are used in C programming?
Ans:-

Basic Data types:
There are four basic types of variable in C; they are: char, int, double, and float.

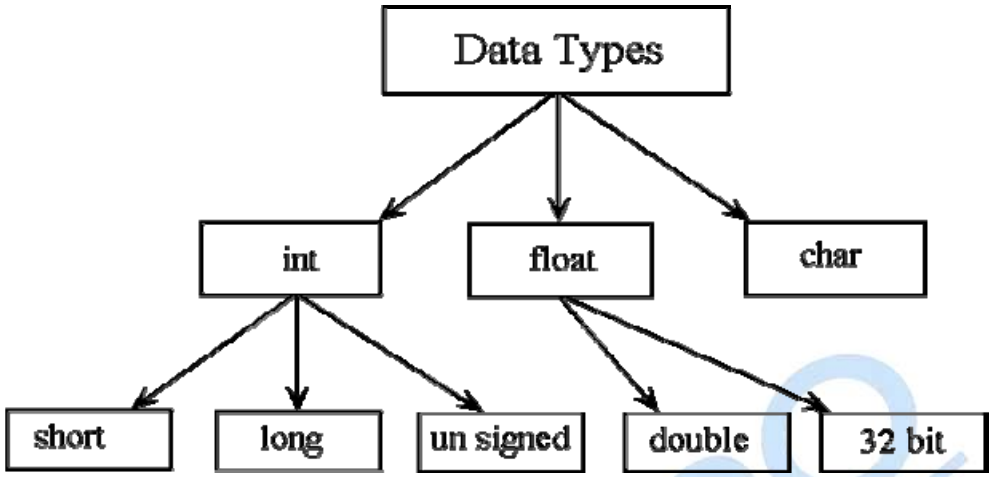
Type name	Meaning
Char	The most basic unit addressable by the machine; typically a single octet(one byte). This is an integer type.
Int	The most natural size of integer for the machine; typically a whole 16, 32 or 64-bit(2, 4, or 8 bytes respectively) addressable word.
Float	A single-precision floating point value.
double	A double-precision floating point value.

To declare a variable of any of these basic types, the name of the type is given first, then the name of the new variable second

```
char red;
int  i;
```

Various qualifiers can be placed on these basic types, in order to further describe their type.

```
unsigned int x;
signed int y;
int z; /* Same as "signed int" */
unsigned char grey;
signed char white;
```



Unsigned integers can only take non-negative values (positive or zero), while signed integers (default) can take both positive and negative values. The advantage of unsigned integers is to allow a greater range of positive values (e.g., 0 → +65535 depending on the size of the integer), whereas signed integers allow only up to half the same number as positive integers and the other half as negative integers (e.g., −32768 → +32767).
An unsigned character, depending on the code page, might access an extended range of characters from 0 → +255, instead of that accessible by a signed char from −128 → +127, or it might simply be used as a small integer. The standard requires char, signed char, unsigned char to be different types. Since most standardized string functions take pointers to plain char, many C compilers correctly complain if one of the other character types is used for strings passed to these functions.
The int type can also be given a size qualifier, to specify more precisely the range of values (and memory size requirements) of the value stored.
short int yellow;
long int orange;
long long int red;
When declaring a short int, long int or long long int, it is permissible to omit the int, as this is implied. The following two declarations are equivalent.
long int brown;
long brown;
The standard is deliberately vague about the sizes of common types. The minimum ranges for common types are given: (Note: On most implementations the maximum negative value for signed types is one higher than the maximum positive, lacking a negative 0)

Type	Minimum allowed range	Typical allowed range	Typical size in bytes
char	−127 → +127 or 0 → +255	−128 → +127 or 0 → +255	1
signed char	−127 → +127	−128 → +127	1
unsigned char	0 → +255	0 → +255	1
signed short int	−32767 → +32767	−32768 → +32767	2
unsigned short int	0 → +65535	0 → +65535	2
signed int	−32767 → +32767	−32768 → +32767 (antique systems) −2147483648 → +2147483647	2 or 4
unsigned int	0 → +65535	0 → +65535 (antique systems) 0 → +4294967295	2 or 4
signed long int	−2147483647 → +2147483647	−2147483648 → +2147483647 −9223372036854775808 → +9223372036854775807 (64-bit systems)	4 or 8
unsigned long int	0 → +4294967295	0 → +4294967295 0 → +18446744073709551615 (64-bit systems)	4 or 8
signed long long int	−9223372036854775807 → +9223372036854775807	−9223372036854775808 → +9223372036854775807	8
unsigned long long int	0 → +18446744073709551615	0 → +18446744073709551615	8
float	1×10−37 → 1×1037	1×10−37 → 1×1037	4
double	1×10−37 → 1×1037	1×10−308 → 1×10308	8
long double	1×10−37 → 1×1037	1×10−308 → 1×10308 1×10−4932 → 1×104932 (x87 FPU systems)	8, 12, or 16

Definitions giving the actual minimum and maximum values of these types can be found in the C standard library header <limits.h> and <float.h>.
The standard does not require that any of these sizes be necessarily different. It is perfectly valid, for example, if all types are 64 bits long. In order to allow a simple and concise description of the sizes a compiler will apply to each of the four types (and the size of a pointer type; see below), a simple naming scheme has been devised; see 64-Bit Programming Models. Two popular schemes are ILP32, in which int, long int and pointer types are 32 bits long; and LP64, in which long int and pointers are 64 bits, and int are 32 bits. Most implementations under these schemes use 16-bit short ints.
A double variable can be marked as being a long double, which the compiler may use to select a larger floating point representation than a plain double. Again, the standard is unspecific on the relative sizes of the floating point values, and only requires a float not to be larger than a double, which should not be larger than a long double.

Q.5: What are operators ? Define different operators those are used in C programming language?

Ans.:- An operator specifies an operation to be performed that yields a value. The variables, constant can be joined by various operators to form an expression. An operand is a data item on which an operator acts. C includes a large number of operator which are as-

1. Arithmetic operators.
2. Relational operators.
3. Logical operators.
4. Assignment operators.
5. Conditional operators.
6. Bitwise operators.
7. Increment or Decrement operators.
8. Special operators.

1. Arithmetic operators :-

Arithmetic operators are used for numeric calculations. They are of two type-

(i.) Unary arithmetic : It requires only one operand. For example :

+b;

Here "+" changes the sign of the operand b.

(ii.) Binary arithmetic : It require two operands. There are five operators for binary arithmetic.

Operator	Meaning	Purpose
+	Plus	Addition
-	Minus	Subtraction
*	Astetisk	Multiplication
/	Slash	Division
%	Modulus	Remainder of integer division

Examples: p=1, q=6, x=5.5, y=2.0; p+q; x+y.

2. **Relational operators**:-Relational operators are used to compare two operands depending on their relations. It requires two operands. These operators are used to compare two values to see whether they are equal, unequal or whether one is greater than the other. An expression that contains the relational operator is called relational expression. If the relation is true it returns the value 1,if the relation is false then it returns the value 0.

Operator	Meaning
<	**Less than**
<=	**Less than or equal**
==	**Equal**
!=	**Not equal**
>	**Greater than**
>=	**Greater than or equal**

3. **Logical or Boolean operator:-** An expression that combines two or more expression is termed as a logical expression. For combining these expression we se the logical or Boolean operator. After testing the value of the condition , it gives the logical status.

Operator	Meaning
&&	**Logical AND**
||	**Logical OR**
!	**Logical NOT**

4. **Assignment operators:-** A value can be stored in variables with use of assignment operator. This assignment operator "=" is used for assignment statement. For example: y=5; Assignment statement is also used for updating statement. For example: x=x+5 can also be return as – x+=15

5. **Ternary or Conditional Operator :-** Ternary operator (? and **:**) requires three operands. This is written as **: Condition ? Operand1 : Operand2 .** First of all the condition is tested and if the result is true then the value of operand1 is taken otherwise value of operand2 is taken. Example :- int minimum; minimum=a < b ? a : b ;

6. **Bitwise operators:-** Chas the ability to supports the manipulation of data at bit level. Bitwise operators are used for operation on bits. Bitwise operators are operated on only integers and characters but not on floats and doubles.

Q.6: Define precedence and associativity of operators in C programming language?

Ans:-Operator precedence
The following is a table that lists the precedence and associativity of all the operators in the C language. Operators are listed top to bottom, in descending precedence. Descending precedence refers to the priority of evaluation. Considering an expression, an operator which is listed on some row will be evaluated prior to any operator that is listed on a row further below it. Operators that are in the same cell (there may be several rows of operators listed in a cell) are evaluated with the same precedence, in the given direction.

A precedence table, while mostly adequate, cannot resolve a few details. In particular, note that the ternary operator allows any arbitrary expression as its middle operand,

despite being listed as having higher precedence than the assignment and comma operators. Thus `a ? b , c : d` is interpreted as `a ? (b, c) : d`, and not as the meaningless `(a ? b), (c : d)`. Also, note that the immediate, unparenthesized result of a C cast expression cannot be the operand of `sizeof`. Therefore, `sizeof (int) * x` is interpreted as `(sizeof(int)) * x` and not `sizeof ((int) *x)`.

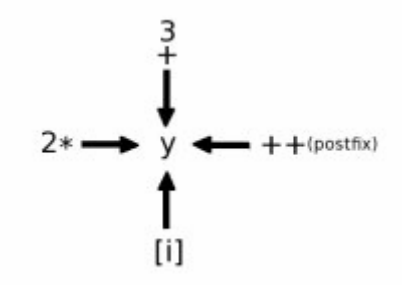
Precedence	Operator	Description	Associativity
1	`::`	Scope resolution (C++ only)	Left-to-right
	`++ --`	Suffix increment and decrement	
	`()`	Function call	
	`[]`	Array subscripting	
	`.`	Element selection by reference	
	`->`	Element selection through pointer	
2	`typeid()`	Run-time type information (C++ only) (see typeid)	
	`const_cast`	Type cast (C++ only) (see const_cast)	
	`dynamic_cast`	Type cast (C++ only) (see dynamic_cast)	
	`reinterpret_cast`	Type cast (C++ only) (see reinterpret_cast)	
	`static_cast`	Type cast (C++ only) (see static_cast)	
	`++ --`	Prefix increment and decrement	Right-to-left
	`+ -`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(type)`	Type cast	
	`*`	Indirection (dereference)	
3	`&`	Address-of	
	`sizeof`	Size-of	
	`new`, `new[]`	Dynamic memory allocation (C++ only)	
	`delete`, `delete[]`	Dynamic memory deallocation (C++ only)	
4	`.* ->*`	Pointer to member (C++ only)	Left-to-right
5	`* / %`	Multiplication, division, and modulus (remainder)	

6	`+ -`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`< <=`	For relational operators < and ≤ respectively	
	`> >=`	For relational operators > and ≥ respectively	
9	`== !=`	For relational = and ≠ respectively	
10	`&`	Bitwise AND	
11	`^`	Bitwise XOR (exclusive or)	
12	`|`	Bitwise OR (inclusive or)	
13	`&&`	Logical AND	
14	`||`	Logical OR	
15	*`c ? t : f`*	Ternary conditional (see ?:)	Right-to-Left
	`=`	Direct assignment (provided by default for C++ classes)	
	`+= -=`	Assignment by sum and difference	
16	`*= /= %=`	Assignment by product, quotient, and remainder	
	`<<= >>=`	Assignment by bitwise left shift and right shift	
	`&= ^= |=`	Assignment by bitwise AND, XOR, and OR	
17	`throw`	Throw operator (exceptions throwing, C++ only)	
18	`,`	Comma	Left-to-right

The precedence table determines the order of binding in chained expressions, when it is not expressly specified by parentheses.

- For example, `++x*3` is ambiguous without some precedence rule(s). The precedence table tells us that: `x` is 'bound' more tightly to `++` than to `*`, so that whatever `++` does (now or later—see below), it does it ONLY to `x` (and not to `x*3`); it is equivalent to (`++x`, `x*3`).
- Similarly, with `3*x++`, where though the post-fix `++` is designed to act AFTER the entire expression is evaluated, the precedence table makes it clear that ONLY `x`

gets incremented (and NOT `3*x`); it is functionally equivalent to something like (`tmp=3*x, x++, tmp`) with `tmp` being a temporary value.



Precedence and bindings

- Abstracting the issue of precedence or binding, consider the diagram above. The compiler's job is to resolve the diagram into an expression, one in which several unary operators (call them 3+(.), 2*(.), (.)++ and (.)[i]) are competing to bind to y. The order of precedence table resolves the final sub-expression they each act upon: (.)[i] acts only on y, (.)++ acts only on y[i], 2*(.) acts only on y[i]++ and 3+(.) acts 'only' on 2*((y[i])++). It's important to note that WHAT sub-expression gets acted on by each operator is clear from the precedence table but WHEN each operator acts is not resolved by the precedence table; in this example, the (.)++ operator acts only on y[i] by the precedence rules but binding levels alone do not indicate the timing of the Suffix ++ (the (.)++ operator acts only after y[i] is evaluated in the expression).

Many of the operators containing multi-character sequences are given "names" built from the operator name of each character. For example, `+=` and `-=` are often called *plus equal(s)* and *minus equal(s)*, instead of the more verbose "assignment by addition" and "assignment by subtraction".

For example, in C, the syntax for a conditional expression is:

```
logical-OR-expression ? expression : conditional-expression
```

Hence, the expression:

```
e = a < d ? a++ : a = d
```

is parsed differently in the two languages. In C, this expression is a syntax error, but many compilers parse it as:

```
e = ((a < d ? a++ : a) = d)
```

which is a semantic error, since the result of the conditional-expression (which might be *a++*) is not an lvalue. In C++, it is parsed as:

```
e = (a < d ? a++ : (a = d))
```

which is a valid expression.

The precedence of the bitwise logical operators has been criticized. Conceptually, & and | are arithmetic operators like + and *.

The expression a & b == 7 is syntactically parsed as a & (b == 7) whereas the expression a + b == 7 is parsed as (a + b) == 7. This requires parentheses to be used more often than they otherwise would.

Q.7: Explain control statements those are used in C programming language?

Ans.: In some situations we may have to change the order of execution of statements based on certain conditions , or repeat a group of statements until certain specified conditions are met. For these situation C provides decision making statements also know as control statements. C has following control statements:

1. if statement.
2. switch statement.
3. Conditional operator.
4. goto statement.

1. if statement:- it is a powerful decision making statement and is used to control the flow of execution of statements. It is basically a two way dscision statement and is used in conjunction with an expression.It takes the following form:

```
if (Text expression)
  {

  };
```

It allows the computer to evaluate the expression first and then, depending on whether the value of the expression is 'true' (or non zero) or 'false' (zero) , it transfer the control to a particular statement. The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are:

1. Simple if statement.

2. if …..else statement.
3. Nested if …..else statement.
4. else if ladder.

2. The switch statement:- C provide multiway decision statement known as a switch.
The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found , a block of statements associated with case is executed. The general form of the switch statement is as shown below:

```
switch(expression)
{
      case value-1:
                  block-1;
                  break;
      case value-2:
                  block-2;
                  break;
     ……
     ……
       default:
                 default- block-;
                  break;
}
Statement-x;
```

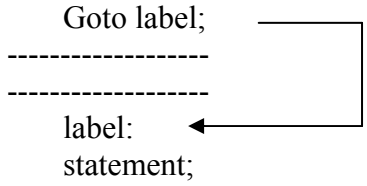
The expression is an integer expression or characters. Value-1,value-2….. are constants or constant expressions and are known as case labels. Block-1,block-2 ….. are statement lists and mat contain zero or more statements. Note that case labels end with a colon(:).
The default is an optional case. When present, it will be executed if the value of the expression does not match with any of case values. if not present, no action takes place if all matches fail and the control goes to the statement-x.

3. The ? : Operator:- The C language has an unusual operator, useful for making two way decisions. This operator is a combination of ? and : ,and takes three operands. This operator is known as the conditional operator. Syntax:-

Conditional expression ? expression1 : expression2

The conditional expression is evaluated first. If the result is nonzero, expression1 is evaluated and is returned as the value of the conditional expression. Otherwise, expression2 is evaluated and its value is returned.

4. The goto statement:- C supports the goto statement to branch unconditionally from one point to another in the program. The goto requires a label in order to identify the branch is to be made. A label I any valid variable name, and must be followed by a colon. The label is placed immediately before the statement where the control is to be transferred. Syntax:



Q.8: Write a program in C programming language to find a largest no in two number?

Ans.:-

```
#include<stdio.h>                          /* Header Files*/
#include<conio.h>
void main()
{
   int a,b;                                /* Declaration*/
   clrscr();
  scanf("%d%d",&a,&b);
  printf("a=%d\nb=%d\n",a,b);
  if (a>b)                                 /* Test expression*/
    {
     printf("\n a is greater than b");
    }
 else
   {
    printf("b  is greater than a");
    }
  getch();
}
```

Q.9: Write a program in C programming language to print weekdays using switch statement?

Ans:-

```
#include<stdio.h>              /* Header Files*/
```

```
#include<conio.h>
void main()
{
int a;                          /* variable declration */
clrscr();
printf("Enter Week day no = ");
scanf("%d",&a);
switch(a)                       /*test expression*/
{  case 1:
          printf("Sunday");
          break;
  case 2:
          printf("Monday");
          break;
  case 3:
          printf("Tuesday");
          break;
  case 4:
          printf("Wednesday");
          break;
  case 5:
          printf("Thrusday");
          break;
  case 6:
          printf("Friday");
          break;
  case 7:
          printf("Staturday");
          break;
  default:
          printf("Wrong Day NO Insert.");
}
getch();

}
```

Q.10: Define iteration Statements?
Or
Define Looping Statements?
Or
Define repeation Statements?

Ans.: C has looping , in which a sequence of statements are executed until some conditions for termination of the loop are satisfied. A program loop therefore consists of two segments, one known as the 'body of the loop' and the other known as the 'control statement' .The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.
Depending on the position of the control of the control statement in the loop, a control structure may be classified either as the 'entry-controlled loop (top down controlled loop) or as the 'exit-controlled loop(bottom up controlled loop)' .The entry controlled and exit controlled loop.
Entry-controlled loops:- while( ) , for () .
Exit controlled loop:- do-while().
C has three forms of iteration statement:

```
do
   <statement>
while ( <expression> ) ;

while ( <expression> )
   <statement>

for ( <expression> ; <expression> ; <expression> )
   <statement>
```

In the while and do statements, the substatement is executed repeatedly so long as the value of the expression remains nonzero (true). With while, the test, including all side effects from the expression, occurs before each execution of the statement; with do, the test follows each iteration. Thus, a do statement always executes its substatement at least once, whereas while may not execute the substatement at all.
If all three expressions are present in a for, the statement

```
for (e1; e2; e3)
   s;
```

is equivalent to

```
e1;
while (e2)
{
   s;
```

```
   e3;
}
```
except for the behavior of a continue; statement (which in the for loop jumps to e3 instead of e2).
Any of the three expressions in the for loop may be omitted. A missing second expression makes the while test always nonzero, creating a potentially infinite loop.
Since C99, the first expression may take the form of a declaration, typically including an initializer, such as

```
for (int i=0; i< limit; i++){
   ...
}
```
The declaration's scope is limited to the extent of the for loop.

Q.11: Write a progam to calculate sum of n integer no.?
Ans.:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,n,sum=0;
printf("enter terms");
scanf("%d",&n);
for(int i=1;i<=n;i++)
{
scanf("%d",&a);
sum=sum+a;
}
printf("\nsum= %d",sum);
getch();
}
```