www.FirstRanker.com

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY: KAKINADA
KAKINADA – 533 003, Andhra Pradesh, India

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

I Year - I Semester		L	T	P	C
		3	0	0	3
PROGRAMMING FOR PROBLEM SOLVING USING C (ES1101)					

COURSE OBJECTIVES:
The objectives of Programming for Problem Solving Using C are

1) To learn about the computer systems, computing environments, developing of a computer program and Structure of a C Program
2) To gain knowledge of the operators, selection, control statements and repetition in C
3) To learn about the design concepts of arrays, strings, enumerated structure and union types. To learn about their usage.
4) To assimilate about pointers, dynamic memory allocation and know the significance of Preprocessor.
5) To assimilate about File I/O and significance of functions

UNIT I
Introduction to Computers: Computer Systems, Computing Environments, Computer languages, Creating and running Programs, Computer Numbering System, Storing Integers, Storing Real Numbers
Introduction to the C Language: Background, C Programs, Identifiers, Types, Variable, Constants, Input/output, Programming Examples, Scope, Storage Classes and Type Qualifiers, Tips and Common Programming Errors Key Terms, Summary, Practice Seat.
Structure of a C Program: Expressions Precedence and Associativity, Side Effects, Evaluating Expressions, Type Conversion Statements, Simple Programs, Command Line Arguments Tips and Common Errors, Key Terms, Summary, Practice Sets.

UNIT II
Bitwise Operators: Exact Size Integer Types, Logical Bitwise Operators, Shift Operators, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Selection & Making Decisions: Logical Data and Operators, Two Way Selection, Multiway Selection, More Standard Functions, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Repetition: Concept of Loop, Pretest and Post-test Loops, Initialization and Updating, Event and Counter Controlled Loops, Loops in C, Other Statements Related to Looping, Looping Applications, Programming Example The Calculator Program, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

UNIT III
Arrays: Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example – Calculate Averages, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Strings: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions String/ Data Conversion, A Programming Example – Morse Code, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Enumerated, Structure, and Union: The Type Definition (Type def), Enumerated Types, Structure, Unions, Programming Application, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY: KAKINADA
KAKINADA – 533 003, Andhra Pradesh, India

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT IV
Pointers: Interdiction, Pointers to pointers, Compatibility, L value and R value, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Pointer Applications: Arrays, and Pointers, Pointer Arithmetic and Arrays, Memory Allocation Function, Array of Pointers, Programming Application, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Processor Commands: Processor Commands, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

UNIT V
Text Input / Output: Files, Streams, Standard Library Input / Output Functions, Formatting Input / Output Functions, Character Input / Output Functions, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Binary Input / Output: Text versus Binary Streams, Standard Library, Functions for Files, Converting File Type, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Functions: Designing, Structured Programs, Function in C, User Defined Functions, Inter-Function Communication, Standard Functions, Passing Array to Functions, Passing Pointers to Functions, Recursion, Passing an Array to Function, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

COURSE OUTCOMES:

The student will
1) Acquires skills to write, compile and debug programs in C language.
2) Be able to use different operators, data types and write programs that use two-way/ multi-way selection.
3) Acquire knowledge to select the best loop construct for a given problem.
4) Design and implements programs to analyze the different pointer applications
5) Design and implements C programs with functions, File I/O operations

TEXT BOOKS:

1. Programming for Problem Solving, Behrouz A. Forouzan, Richard F.Gilberg, CENGAGE
2. The C Programming Language, Brian W.Kernighan, Dennis M. Ritchie, 2e, Pearson
3. Programming in C, Reema Thareja, OXFORD

REFERENCE:

1. Computer Fundamentals and Programming, Sumithabha Das, Mc Graw Hill
2. Programming in C, Ashok N. Kamthane, Amit Kamthane, Pearson
3. Computer Fundamentals and Programming in C, Pradip Dey, Manas Ghosh, OXFORD

UNIT-1
INTRODUCTION TO COMPUTERS

What is a Computer?

The word “Computer” comes from the word “compute” which means to calculate. A computer is an electronic device, which stores and processes data to give meaningful information. Processing is done with the help of instructions given by the user, which are also stored within the computer.
Data refers to all the basic elements that can be produced or processed by a computer.
Data is a collection of facts and figures which has to be processed by some processing system, (whether a human being or a machine) to be understandable.
Information: It is the processed form of data, which makes some sense and helps in reaching aconclusion.

Characteristics of Computers

- Speed
- Storage
- Logical decision
- Super Efficiency and Automation
- Accuracy
- Reliability
- Versatility

$$\begin{tikzcd}\boxed{\textbf{INPUT}} \arrow[r] & \boxed{\textbf{PROCESS}} \arrow[r] & \boxed{\textbf{OUTPUT}}\end{tikzcd}$$

Computer system is made up of the following components

- Input device
- Central ProcessingUnit
- Output device

Input device
Input means putting the raw data through input device into the processing device Input Unit allows us to communicate with the computer. An input unit converts the numbers, alphabet or other signals into the internal binary code (will be discussed later). Keyboard, mouse, joystick, light pen are the examples of input unit. We will discuss about the input device in details later.

Central Processing Unit
Another important component of the computer system is known as CPU. It accepts instructions through keyboard, stores them into memory and later on executes them. It can be also thought of as the "brain" of the computer. It performs all the calculations and controls the overall activities of the computer.

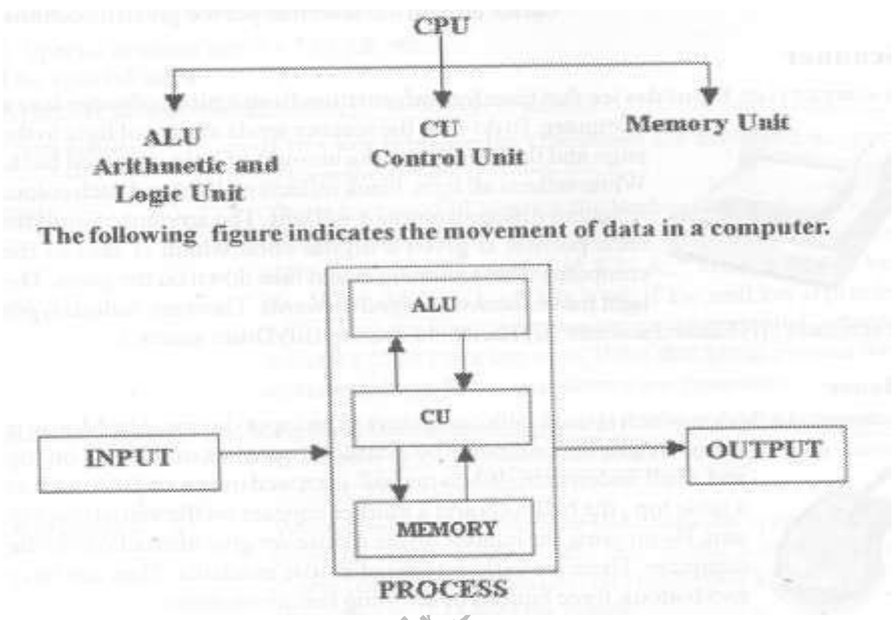
CPU consists of three components

- Arithmetic LogicUnit
- MemoryUnit
- Control Unit

Output device
The devices, through which we get the processed information or the desired result, are known as the output devices. There are a variety of output devices which return the information given by the user. The most popularly used output devices are Visual Display Unit (VDU) or monitor, printers, plottersetc.

Central Processing Unit: Central Processing Unit (CPU) is the processing device of computer. It inhere that the actual work is done. The CPU is like the `brain' of the computer. It takes information from the input unit and memory and uses or processes it according to the instructions given. It may put information into the memory or give results to the output unit.

The following figure indicates the movement of data in a computer.



ALU (Arithmetic and Logic Unit) In the ALU, calculations such as addition, multiplication etc. of numbers is carried out. The ALU can also carry out logicaloperations like comparing two numbers to see which is the larger. A modern computer can perform a single addition in nanoseconds (1 nano second = 10^{-9}seconds)

Control Unit: It controls the overall activities of a computer system. It getsinformation from the input unit, sends information to the output unit or transfers information to or from the computer's memory. It is important that everything is done in exactly the right order and at the right time, so there is an accurate clock within the computer which is connected to the control unit.

Internal Memory / Primary Memory Ina PC, internal memory is usually contained insilicon chips. It holds instructions and data which the computer is currently working on, and which can be accessed by the CPU whenever required. Internalmemory needs to work very rapidly because the speed of the CPU is very high and requires information to be readilyavailable.

Internal Memory is of two types
RAM ROM

RAM (Random Access Memory) is an electronic memory which is used to storedata and instructions from the operating system and any program you are using. RAM Storesinformation temporarily.If the power is interrupted, even for an instant the information is lost forever. The CPU accepts information from RAM as and when required, processes it and returns toRAM.

ROM (Read Only Memory) ROM or Read Only Memory holds sets of instructions which tell the computer what to do. For instance, a ROM will tell the processor how to recognize which key has been pressed and how to light up the screen. Information stored in ROM can be "read" it can notbe erased or added to because when chip is manufactured it is made non-writable. The information stored in ROM is not erased even if Power is switchedoff.

Disk Drives

Disk drives provide a means of storing work, or data. *Floppy disks* are transportable from PC to PC and come in two sizes, 31/2" and 51/4" diameter. *Hard disks* are fixed inside the system unit and have much higher storage capacities thanfloppies.

Hard disks (or fixed disks) work on the same principle as floppy disks but are fixed inside the PC in a sealed unit. They can store a great deal more information than floppy disks and range in capacity from 10MB to several hundred MB. Access times (i.e. the time taken to read and write information) for hard disks are much faster than for floppy disks. Manufacturers often quote access times as well as capacities for harddisks.

Information is stored on disk in the form of *files*. A file might be a program or data such as a word processor document. Files can be grouped together on disk in *directories*.

The Monitor

The monitor provides display output from the PC. Monitors vary in screen resolution and colours available. Monitors are available in colour and monochrome versions and in different screensizes.

The Keyboard

The keyboard allows you to input commands and information into the PC. The keyboard is normally connected to the main unit via a 5 pin DIN type socket.

Peripherals
As well as the essential keyboard and monitor, *peripheral* items such as printers and mice are often found connected to PCs.

Printers and Plotters
A printer may be connected to one of the serial or, more commonly, parallel ports of a PC. The availability of a printer is especially important for applications such as word processing. Printers vary enormously in quality and speed ofoutput.

Printer Types
Printers can be grouped by the method with which they print.

- Dot-Matrix Printers
- Laser Printers
- Inkjet Printers
- Plotters

The Mouse
A mouse is a device which moves a pointer around the screen, options being selected by pressing (or `clicking') a mouse button. In present day Software a mouse is essential. Graphical User Interfaces, such as Windows require the use of a mouse.

The Keyboard
This section covers the use of the keyboard.

Layout of the Keyboard

There are two styles of keyboard in common use, the older 84-key and the now more common 102-key expanded keyboard. The 102-key keyboard normally consists of four main parts;

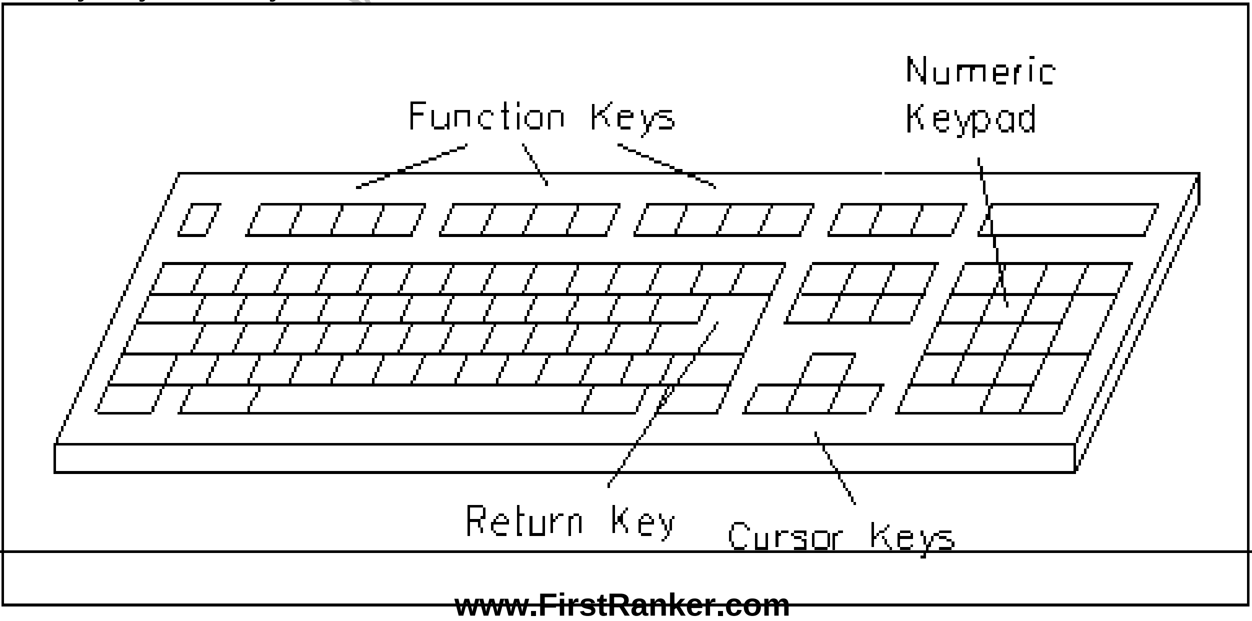
A typewriter-style alphanumeric keyboard with four extra keys labeled Esc (Escape), Ctrl (Control), and Alt (Alternate).

A numeric keypad, situated at the far right of the keyboard, providing the digits 0 to 9, a decimal point and plus and minus signs, together with four special keys labeled Num Lock (Number Lock), Scroll Lock, Ins (Insert) and Del (Delete). On the 84-key keyboard the numeric keys are also used as cursor control keys. In between the alphabetic keys and the numeric keypad there are two groups of keys. The top group repeats the functions on the numeric keypad. Below there are four cursor keys. These are not present on the 84-keykeyboard.

A set of twelve *function* keys situated in a single row above the main typewriter keyboard and labeled F1 to F12. On the 84-key keyboards there are ten function keys situated to the left of the main typewriter keys.

The layout of these keys is shown in figure and their general uses are described in the following sections. The position of some of the keys may be different from that describedbelow.

102 Key Keyboard Layout



What is Hardware ?

Hardware is the physical part of a computer, something which we can touch, feels with our hand. Technically, we can define the Hardware as all the equipment and electronic circuits that make up the computer i.e. keyboard, screen, disk drives, printers etc. However the hardware can do nothing without the software

What is Software?

Software is the set of operational instructions to the hardware, which tells the computer what to do, how to act, how to generate picture, how to print a bio-data and other documents and so on. Technically we can define the software as the information that the computer needs to work on. The information can be instructions, which tell the computer what to do or the data that is used by the instruction. For example to perform addition, the actual numbers that are entered into the computer for addition are the data.

A set of instructions that performs a task is known as a program. You cannot touch the software; it can only be stored on floppy disk, hard disk drive, Compact Disks (CDs), just the way music is stored in cassettes and CDs.

Types of Software

System Software: this is the most important software which is used as an interpreter between the user and the computer. It also manages the input and output devices of a computer.

Application Software: this is customized software created for user's specific need and is done with the help of a programming language.

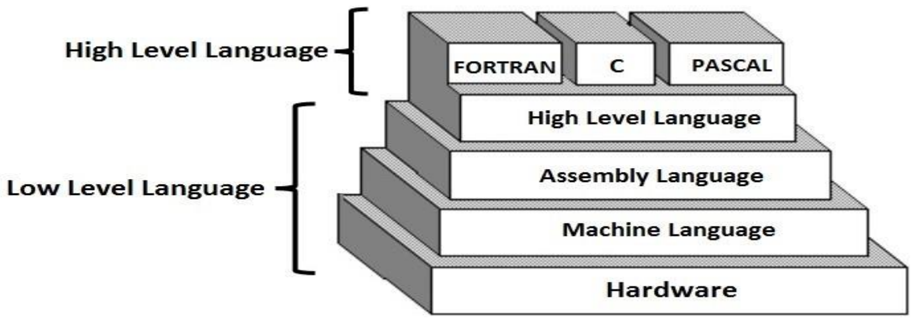
Computer LANGUAGES

Introduction:

A language is the main medium of communicating between the Computer systems and the most common are the programming languages. As we know a Computer only understands binary numbers that is 0 and 1 to perform various operations but the languages are developed for different types of work on a Computer. A language consists of all the instructions to make a request to the system for processing a task. Here we will go in the detail of the Computer language and its types.

Computer Language Description:

A Computer language includes various languages that are used to communicate with a Computer machine. Some of the languages like programming language which is a set of codes or instructions used for communicating the machine. Machine code is also considered as a computer language that can be used for programming. But all the languages that are now available are categorized into two basic types of languages including Low-level language and High level language.



Computer Language and its Types

Low Level Language:

Low level languages are the machine codes in which the instructions are given in machine language in the form of 0 and 1 to a Computer system. It is mainly designed to operate and handle all the hardware. The main function of the Low level language is to operate, manage and manipulate the hardware and system components. There are various programs and applications written in low level languages that are directly executable without any interpretation or translation. Low level language is also divided into two parts are Machine language and Assembly language.

- Machine Language is one of the low-level programming languages which is the first generation language developed for communicating with a Computer. It is written in machine code which represents 0 and 1 binary digits inside the Computer string which makes it easy to understand and perform the operations. As we know a Computer system can recognize electric signals so here 0 stands for turning off electric pulse and 1 stands for turning on electric pulse. It is very easy to understand by the Computer and also increases the processing speed.

The main advantage of using Machine language is that there is no need of a translator or interpreter to translate the code, as the Computer directly can understand. But there are some disadvantages also like you have to remember the operation codes, memory address every time you write a program and also hard to find errors in a written program. It is a machine dependent and can be used by a single type of Computer.

- Assembly Language is the second generation programming language that has almost similar structure and set of commands as Machine language. Instead of using numbers like in Machine languages here we use words or names in English forms and also symbols. The programs that have been written using words, names and symbols in assembly language are converted to machine language using an Assembler. Because a Computer only understands machine code languages that's why we need an Assembler that can convert the Assembly level language to Machine language so the Computer gets the instruction and responds quickly.

The main disadvantage of this language is that it is written only for a single type of CPU and does not run on any other CPU. But its speed makes it the most used low level language till today which is used by many programmers.

High Level Language:

The high level languages are the most used and also more considered programming languages that helps a programmer to read, write and maintain. It is also the third generation language that is used and also running till now by many programmers. They are less independent to a particular type of Computer and also require a translator that can convert the high level language to machine language. The translator may be an interpreter and Compiler that helps to convert into binary code for a Computer to understand. There is various high level programming languages like C, FORTRAN or Pascal that are less independent and also enables the programmer to write a program.

The Compiler plays an important role on the Computer as it can convert to machine language and also checks for errors if any before executing. There are several high level languages that were used earlier and also now like COBOL, FORTRAN, BASIC, C, C++, PASCAL, LISP, Ada, Algol, Prolog and Java. It is user-friendly as the programs are written in English using words, symbols, characters, numbers that needs to be converted to machine code for processing.

Step By Step Execution Program

Step 1 : Edit

1. This is First Step i.e **Creating a Program**.
2. First Write Program using Text Editor
3. Save Program by using Extension.
4. File Saved with extension is called "**Source Program**".

Step 2 : Compiling

1. Compiling C Program : C Source code with Extension is given as input to compiler and compiler convert it into Equivalent Machine Instruction.
2. In Borland C/C++ 3.0 program can be compiled using key **[Alt + F9]**.
3. **Compiler Checks for errors** . If source code is error-free then Code is converted into Object File [.Obj].

Step 3 : Checking Errors

1. During **Compilation Compiler will check for error**, If compiler finds any error then it will report it.
2. User have to **re-edit the program**.
3. After **re-editing program , Compiler again check for any error**.
4. If program is error-free then program is linked with appropriate libraries.

Step 4 : Linking Libraries

1. Program is linked with **included header files**.
2. Program is linked with other libraries.
3. This process is **executed by Linker**.

Step 5 :Getting Result

1. After all the above steps are completed press ALT+F5.
2. Then the result is displayed on the output screen

Number System:

When we type some letters or words, the computer translates them in numbers as computers can understand only numbers. A computer can understand the positional number system where there are only a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

Decimal Number System

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands, and so on.

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position. Its value can be written as

(1 x 1000)+ (2 x 100)+ (3 x 10)+ (4 x l)
$(1 \times 10^3)+ (2 \times 10^2)+ (3 \times 10^1)+ (4 \times 10^0)$
1000 + 200 + 30 + 4
1234

S.No.	Number System and Description
1	**Binary Number System** Base 2. Digits used : 0, 1
2	**Octal Number System** Base 8. Digits used : 0 to 7
3	**Hexa Decimal Number System** Base 16. Digits used: 0 to 9, Letters used : A- F

Binary Number System

Characteristics of the binary number system are as follows −

- Uses two digits, 0 and 1

- Also called as base 2 number system
- Each position in a binary number represents a **0** power of the base (2). Example 2^0
- Last position in a binary number represents a **x** power of the base (2). Example 2^x where **x** represents the last position - 1.

Example:-

Binary Number: 10101_2

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	10101_2	$((1 \text{ x } 2^4) + (0 \text{ x } 2^3) + (1 \text{ x } 2^2) + (0 \text{ x } 2^1) + (1 \text{ x } 2^0))_{10}$
Step 2	10101_2	$(16 + 0 + 4 + 0 + 1)_{10}$
Step 3	10101_2	21_{10}

Note − 10101_2 is normally written as 10101.

Octal Number System

Characteristics of the octal number system are as follows −

- Uses eight digits, 0,1,2,3,4,5,6,7
- Also called as base 8 number system
- Each position in an octal number represents a **0** power of the base (8). Example 8^0
- Last position in an octal number represents a **x** power of the base (8). Example 8^x where **x** represents the last position – 1

Example

Octal Number: 12570_8

Calculating Decimal Equivalent −

Step	Octal Number	Decimal Number
Step 1	12570_8	$((1 \text{ x } 8^4) + (2 \text{ x } 8^3) + (5 \text{ x } 8^2) + (7 \text{ x } 8^1) + (0 \text{ x } 8^0))_{10}$
Step 2	12570_8	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570_8	5496_{10}

Note − 12570_8 is normally written as 12570.

Hexadecimal Number System

Characteristics of hexadecimal number system are as follows −

- Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Letters represent the numbers starting from 10. A = 10. B = 11, C = 12, D = 13, E = 14, F = 15
- Also called as base 16 number system
- Each position in a hexadecimal number represents a **0** power of the base (16). Example, 16^0
- Last position in a hexadecimal number represents a **x** power of the base (16). Example 16^x where **x** represents the last position - 1

Example

Hexadecimal Number: $19FDE_{16}$

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536+ 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

Introduction to C Language:

C language facilitates a very efficient approach to the development and implementation of computer programs. The History of C started in 1972 at the Bell Laboratories, USA where Dennis M. Ritchie proposed this language. In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce “an unambiguous and machine independent definition of the language C while still retaining it’s spirit .

C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C. Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on. Machine-specific features are isolated in a few modules within the UNIX kernel, which makes it easy for you to modify them when you are porting to different hardware architecture.

C was first designed by Dennis Ritchie for use with UNIX on DEC PDP-11 computers. The language evolved from Martin Richard's BCPL, and one of its earlier forms was the B language, which was written by Ken Thompson for the DEC PDP-7. The first book on C was The C Programming Language by Brian Kernighan and Dennis Ritchie, published in 1978.

In 1983, the American National Standards Institute (ANSI) established a committee to standardize the definition of C. The resulting standard is known as ANSI C, and it is the recognized standard for the language, grammar, and a core set of libraries. The syntax is slightly different from the original C language, which is frequently called K&R for Kernighan and Ritchie. There is also an ISO (International Standards Organization) standard that is very similar to the ANSI standard.

It appears that there will be yet another ANSI C standard officially dated 1999 or in the early 2000 years; it is currently known as "C9X."

IDENTIFIERS :

Names of the variables and other program elements such as functions, array,etc,are known as identifiers.

There are few rules that govern the way variable are named(identifiers).

1. Identifiers can be named from the combination of A-Z, a-z, 0-9, _(Underscore).

2. The first alphabet of the identifier should be either an alphabet or an underscore. digit are not allowed.
3. It should not be a keyword. Eg: name,ptr,sum

After naming a variable we need to declare it to compiler of what data type it is . The format of declaring a variable is

Data-type id1, id2,. idn;

where data type could be float, int, char or any of the data types.

id1, id2, id3 are the names of variable we use. In case of single variable no commas are required.

eg float a, b, c;

int e, f, grand total; char present_or_absent;

DATA TYPES :

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. C supports four different classes of data types namely

1. Basic Data types
2. Derives data types
3. User defined data types
4. Pointer data types

BASIC DATA TYPES:

All arithmetic operations such as Addition , subtraction etc are possible on basic data types.

E.g.: int a,b;
Char c;

The following table shows the Storage size and Range of basic data types:

TYPE	**LENGTH**	**RANGE**
Unsigned char	8 bits	0 to 255
Char	8 bits	-128 to 127
Short int	16 bits	-32768 to 32767
Unsigned int	32 bits	0 to 4,294,967,295
Int	32 bits	-2,147,483,648 to 2,147,483,648
Unsigned long	32 bits	0 to 4,294,967,295
Enum	16 bits	-2,147,483,648 to 2,147,483,648
Long	32 bits	-2,147,483,648 to 2,147,483,648
Float	32 bits	3.4*10E-38 to 3.4*10E38
Double	64 bits	1.7*10E-308 to 1.7*10E308
Long double	80 bits	3.4*10E-4932 to 1.1*10E4932

DERIVED DATA TYPES:

Derived datatypes are used in 'C' to store a set of data values. Arrays and Structures are examples for derived data types.

Ex: int a[10];
Char name[20];

USER DEFINED DATATYPES:

C Provides a facility called typedef for creating new data type names defined by the user. For Example , the declaration ,**typedef int Integer;**makes the name Integer a synonym of int. Now the type Integer can be used in declarations, casts,etc,like,

Integer num1,num2;

Which will be treated by the C compiler as the declaration of num1,num2as int variables. "typedef" ia more useful with structures and pointers.

POINTER DATA TYPES:

Pointer data type is necessary to store the address of a variable.

VARIABLE:

A **variable** in simple terms is a storage place which has some memory allocated to it. Basically, a variable used to store some form of data. Different types of variables require different amounts of memory, and have some specific set of operations which can be applied on them.

Variable Declaration:

A typical variable declaration is of the form:

```
type variable_name;
   or for multiple variables:
 type variable1_name, variable2_name, variable3_name;
```

Constants

If you want to define a variable whose value cannot be changed, you can use the `const` keyword. This will create a constant. For example,

```
        const double PI = 3.14;
```

Notice, we have added keyword `const`.

Here, `PI` is a symbolic constant; its value cannot be changed.

```
        const double PI = 3.14;
        PI = 2.9; //Error
```

Scope of a VARIABLE:

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language −

- Inside a function or a block which is called **local** variables.
- Outside of all functions which is called **global** variables.

Let us understand what are **local** and **global** variables.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. The following example shows how local variables are used. Here all the variables a, b, and c are local to main() function.

Global Variables

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. The following program show how global variables are used in a program.

STORAGE CLASSES IN C:-

Type refers to the data type of a variable. And, storage class determines the scope, visibility and lifetime of a variable.

There are 4 types of storage class:

1. Automatic
2. External
3. Static
4. register

Local Variable

The variables declared inside a block are automatic or local variables. The local variables exist only inside the block in which it is declared.

Let's take an example.

```
#include <stdio.h>

int main(void) {

 for (int i = 0; i < 5; ++i) {
   printf("C programming");
 }

// Error: i is not declared at this point
 printf("%d", i);
 return 0;
}
```

When you run the above program, you will get an error undeclared identifier i. It's because i is declared inside the for loop block. Outside of the block, it's undeclared.

Let's take another example.

```
int main() {
    int n1; // n1 is a local variable to main()
}

void func() {
   int n2;  // n2 is a local variable to func()
}
```

In the above example, n1 is local to main() and n2 is local to func().
This means you cannot access the n1 variable inside func() as it only exists inside main(). Similarly, you cannot access the n2 variable inside main() as it only exists inside func().

Global Variable

Variables that are declared outside of all functions are known as external or global variables. They are accessible from any function inside the program.

Example 1: Global Variable

```
#include <stdio.h>
void display();

int n = 5;  // global variable

int main()
{
    ++n;
    display();
    return 0;
}

void display()
```

```
{
   ++n;
   printf("n = %d", n);
}
```

Output

```
n = 7
```

Register Variable

The `register` keyword is used to declare register variables. Register variables were supposed to be faster than local variables.

However, modern compilers are very good at code optimization, and there is a rare chance that using register variables will make your program faster.

Unless you are working on embedded systems where you know how to optimize code for the given application, there is no use of register variables.

Static Variable

A static variable is declared by using the `static` keyword. For example;

```
static int i;
```

The value of a static variable persists until the end of the program.

Example 2: Static Variable

```
#include <stdio.h>
void display();

int main()
{
   display();
   display();
}
void display()
{
   static int c = 1;
   c += 5;
   printf("%d  ",c);
}
```

TYPES OF C TYPE QUALIFIERS:

There are two types of qualifiers available in C language. They are,

1. const
2. volatile

1. CONST KEYWORD:

1. Constants are also like normal variables. But, only difference is, their values can't be modified by the program once they are defined.
2. They refer to fixed values. They are also called as literals.
3. They may be belonging to any of the data type.

Syntax:
const data_type variable_name; (or) const data_type *variable_name;
Please refer **C – Constants** topic in this tutorial for more details on const keyword.

2. VOLATILE KEYWORD:

1. When a variable is defined as volatile, the program may not change the value of the variable explicitly.
2. But, these variable values might keep on changing without any explicit assignment by the program. These types of qualifiers are called volatile.
3. For example, if global variable's address is passed to clock routine of the operating system to store the system time, the value in this address keep on changing without any assignment by the program. These variables are named as volatile variable.

Syntax:
volatile data_type variable_name; (or) volatile data_type *variable_name;

Operator Precedence and Associatively in C

Operator precedence: It dictates the order of evaluation of operators in an expression.
Associativity: It defines the order in which operators of the same precedence are evaluated in an expression. Associativity can be either from left to right or right to left.
Consider the following example:

1 24 + 5 * 4

Here we have two operators + and *, Which operation do you think will be evaluated first, addition or multiplication? If the addition is applied first then answer will be 116 and if the multiplication is applied first answer will be 44. To answer such question we need to consult the operator precedence table.

In C, each operator has a fixed priority or precedence in relation to other operators. As a result, the operator with higher precedence is evaluated before the operator with lower precedence. Operators that appear in the same group have the same precedence. The following table lists operator precedence and associativity.

Precedence and associatively table:-

OPERATOR	DESCRIPTION	ASSOCIATIVITY
( ) [ ] . -> ++ —	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2)	left-to-right
++ — + – ! ~ (*type*) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of *type*) Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right

<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

TYPE CASTING

Type casting is a way to convert a variable from one data type to another data type. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'. You can convert the values from one type to another explicitly using the **cast operator** as follows −

```
(type_name) expression
```

Consider the following example where the cast operator causes the division of one integer variable by another to be performed as a floating-point operation.

```
#include <stdio.h>

main() {

  int sum = 17, count = 5;
  double mean;

  mean = (double) sum / count;
  printf("Value of mean : %f\n", mean );
}
```

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

```
Value of mean : 3.400000
```

It should be noted here that the cast operator has precedence over division, so the value of **sum** is first converted to type **double** and finally it gets divided by count yielding a double value.

Type conversions can be implicit which is performed by the compiler automatically, or it can be specified explicitly through the use of the **cast operator**. It is considered good programming practice to use the cast operator whenever type conversions are necessary.

Integer Promotion

Integer promotion is the process by which values of integer type "smaller" than **int** or **unsigned int** are converted either to **int** or **unsigned int**. Consider an example of adding a character with an integer −

```
#include <stdio.h>

main() {

  int  i = 17;
  char c = 'c'; /* ascii value is 99 */
  int sum;

  sum = i + c;
  printf("Value of sum : %d\n", sum );
}
```

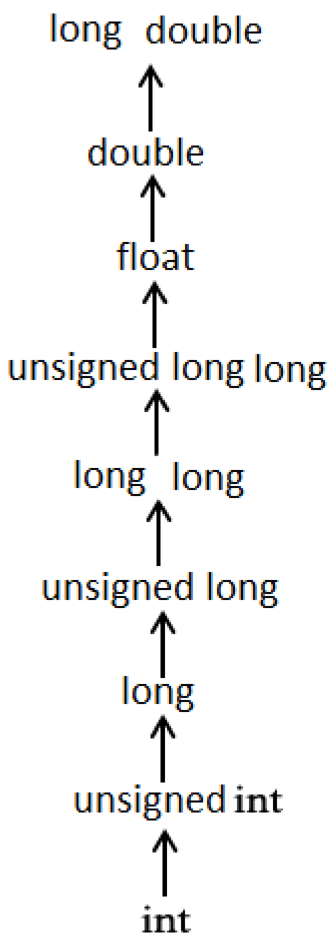
When the above code is compiled and executed, it produces the following result −

```
Value of sum : 116
```

Here, the value of sum is 116 because the compiler is doing integer promotion and converting the value of 'c' to ASCII before performing the actual addition operation.

Usual Arithmetic Conversion

The **usual arithmetic conversions** are implicitly performed to cast their values to a common type. The compiler first performs *integer promotion*; if the operands still have different types, then they are converted to the type that appears highest in the following hierarchy −



The usual arithmetic conversions are not performed for the assignment operators, nor for the logical operators && and ||. Let us take the following example to understand the concept

```
#include <stdio.h>

main() {

   int  i = 17;
   char c = 'c'; /* ascii value is 99 */
   float sum;

   sum = i + c;
   printf("Value of sum : %f\n", sum );
}
```

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

```
Value of sum : 116.000000
```

Here, it is simple to understand that first c gets converted to integer, but as the final value is double, usual arithmetic conversion applies and the compiler converts i and c into 'float' and adds them yielding a 'float' result.

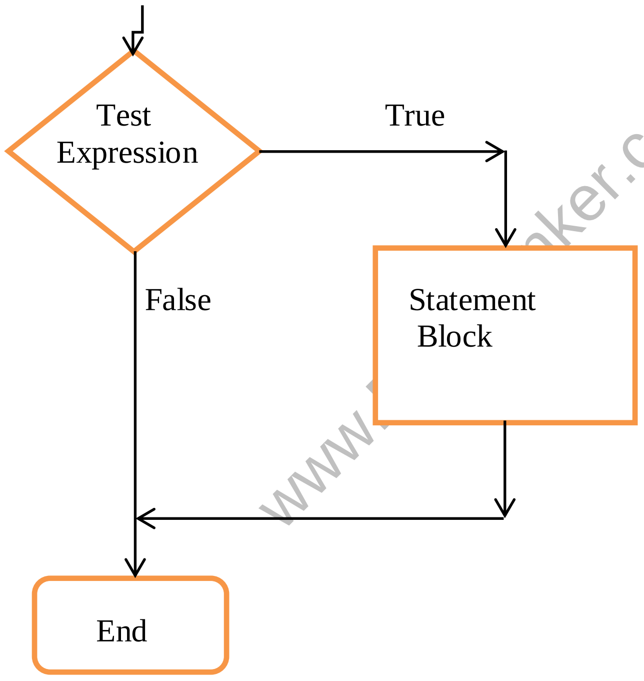
UNIT-II
DECISION MAKING STATEMENTS

Decision making with 'if' statements:

The **if** Statement is a powerful decision making statement and is used to control the flow of execution of statements.it is basically a two-way decision statement and is used in conjunction with an expression.
They are 3 syntax of **if** statements

- ✓ Simple if
- ✓ If else
- ✓ And nested

Simple if:
```
If(test expression)
{
        Statement-block;
}
```
The statement-block may be a single statement or a group of statements. If the expression is true the statement-block will be executed. Otherwise the statement-block will be skipped and the execution will jump after the closing brace.



//program to grade of mark//

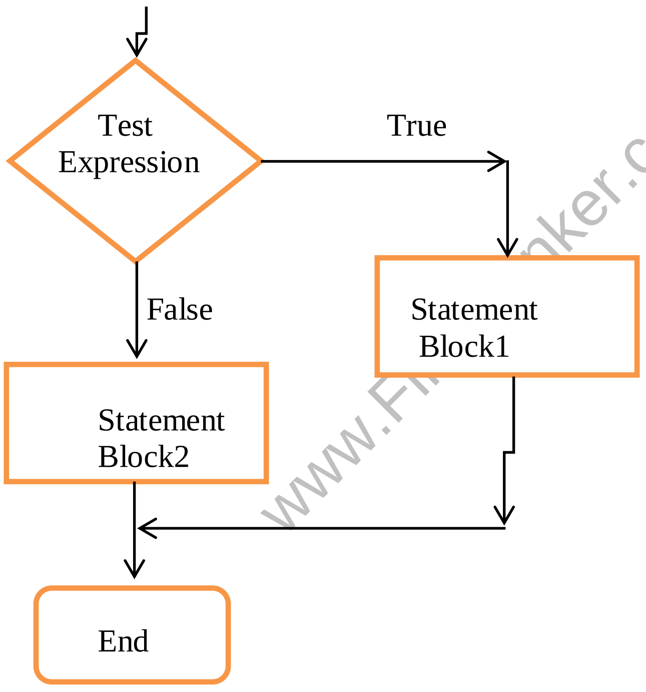
#include<stdio.h>

```
int main()
{
        int marks;
        printf("\n enter marks");
        scanf("%d",&marks");
        if(marks>=90)
        printf("Grade A");
return 0;
}
```

Syntax of if else:

The general form of if-else statement:

```
if(expression)
{
        Statement_block(1);
}
else
{
        Statement_block(2);
}
```



If the expression is true statement-block1 will be executed.Otherwise statement block2 will Be executed. In both cases control is transferred subsequent o statements after if statement.

//program to find leap year

```
#include<stdio.h>
int main()
{
        int year;
        printf("enter year");
        scanf("%d",&year);
        if(year%4==0)
              printf("leap year");
        else
              printf("not leap year");
        return0;
}
```

Nested of if-else
When a series of decision are involved, we may have to use more than one if-else statement in nested form.
Syntax:

```
if(cond-1)
  Stmt1;
elseif(cond-2)
           Stmt2;
elseif(cond-3)
            Stmt3;
     else
            Stmt4;
```

//Example program to print grades

```
#include<stdio.h>
int main()
{
        int marks;
        printf("enter marks");
        scanf("%d",&marks);
        if(marks>=90)
                printf("gradeA");
        else if(marks>=80)
                printf("Grade B);
        else if(marks>=70)
                printf("Grade C");
            else
                printf("Fail");
        return 0;
}
```

//Example program to print Grades using logical operators &&

```
#include<stdio.h>
int main()
{
        int marks;
        printf("enter marks");
        Scanf("%d",&marks);
        if((marks>=90)&&(marks<=100)
                Printf("Grade A");
        if((marks>=80)&&(marks<90))
                Printf("GradeB");
        if((marks>=70)&&(mark<80)
                printf("Grade C");
        return 0;
}
```

//Exmple program to print Grades using logical operators ||

```
#include<stdio.h>
int main()
{
        char op;
        printf("\n enter y for yes and n for No");
        scanf("%c",&op);
        if(op=='Y'||op='y')
                printf("the selected option is ye");
        if(op=='N'||op='n')
                printf("The selected option is No");
        return 0;
}
```

The switch Statement:

When one of the many alternatives is to be selected,we can design a program using if statements to control the selection.C has a built-in multi way decision statement known as '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 that caseis executed.

Syntax:

```
switch(expression)
{
case value1:   block-1;
              break;
case value2:   block-2;
               break;
……….
default:       default-block;
               break;
}
```

The expression is an integer expression or characters value1, value2…… are constants and are known as case labels. Each of these values should be unique within a switch statement.block1, block2…. Are statements lists and may contain zero or more statements.There is noneed to put

braces around these blocks .case labels end with colon(:). When the switchis executed, the value of the expression is successively compared against the values value-1,Value-2….. if a case is found whose value matches with the value of the expression,then theblock of statements that follow the case are executed.
The break statement at the end of each block signal the end of a particular case and causesan exit from the witch statement,transferring the control to the statements following the switch.The default is an optional case.when present,it will be executed if the value of theexpression does not match with any of these case values.if not present , no action takesplace if all matches fail and control goes to the next statement of switch.

/*Program to operate switch statement*/

```
#include<stdio.h>
#include<conio.h>
main()
{
        int a,b,sum,diff,product,division;
        char op;
        clrscr();
        Printf("enter two operands");
        Scanf("%d%d",&a,&b);
        printf("operator + - * / ? ");
        fflush(stdin);
        scanf("%c",&op);
        switch(op)
        {
        case '+':sum=a+b;
                Printf("sum=%d\n",sum);
                break;
        case '-' : diff=a-b;
                printf("difference=%d\n",diff);
                break;
        case '*': product=a*b;
                printf("product=%\n",product);
                break;
        case '/': division=a/b;
                printf("division=%d\n",product);
                break;
        default: printf("error");
                break;
        }
getch();
}
```

Goto Statement:
The goto statement is used to branch unconditionally from one point to another in the program.The goto requires a label in order to identify the place where the branch is tobemade.The label is made immediately before the statement where the control is tobetransferred.

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

The label: can be anywhere in the program either or after the goto label.

Loops:

There may be a situation, when we need to execute a block of code several number of times. In general, statements are executed sequentially. i.e., The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of loop statements in most of the programming languages.

There are three forms of loop statements:

- ✓ for loop
- ✓ while loop
- ✓ do loop

The for loop

Syntax:

```
for(initialization; test condition; update expression)
{
  statements ;
}
```

- ✓ initialization is an expression with side –effect that initializes a control variable (typically an assignment), which can also be a declaration with initialization
- ✓ Condition is an expression of type Boolean
- ✓ Update is an expression with side –effect that typically consists in updating(i.e., increamenting or decrementing) the control variable
- ✓ Statement is a single statement (also called body of the loop)

Example : To print out integers up to 100.

```
for(i = 0 ;i< 100; i++)
printf("%d",i);
```

The while statement

It allows for the repetition of a statement.

Syntax:

```
While (Test condition )
{
    Statement  ;
}
```

- Condition is an expression of type Boolean
- Statement is a single statement (also called the body of the loop)

Since, by making use of a block, it is possible to group several statements into a single composite statement, it is in fact possible to have more than one statement in the body of the loop.

Execution of loop:

- First,the condition is evaluated.
- If it is true, the statement is executed and the value of the condition is evaluated again,continuing in this way until the condition becomes false.
- At this point the statement immediately following the while loop is executed.

Hence , the body of the loop is executed as the condition says true . As soon as it becomes false we exit the loop and continue with the following statement.

Example : print 100 stars.

```
int i =0;
while (i<100)
{
        printf("*");
        i++;
}
```

The do loop

In a while loop, the condition of end of loop is checked at the beginning of each iteration.
A do loop is similar to a while loop,with the only difference that the condition of end of loop is checked at the end of each iteration.

Syntax :

```
do
{
  statement
}while (condition);
```

- ✓ condition is an expression of type bollean
- ✓ statement is a single statement (also called the body of the loop)

Execution:

- ✓ First the statement is executed.
- ✓ Then the condition is evaluated, and if it is true , the statement is executed again, continuing in this way until the condition becomes false.
- ✓ At this point the statement immediately following the do loop is executed.

Example: To print out integers up to 100.

```
int = 0;
```

```
do {
        printf (“%d”, i);
        i++;
}while(i< 100);
```

/* program to find factorial of a given integer using WHILE LOOP*/

```
#include<stdio.h>
int main ()
{
        int n, fact, i;
        printf(“enter an integer”);
        scanf(“%d”, &n);
        fact=1;
        i=1;
        while(i<=n)
        {
            fact =fact*i;
             i++;
        }
        printf(“factorial of %d is %d\n”,n,fact);
return 0;
}
```

/*program to find sum n natural numbers using for loop*/

```
#include<stdio.h>
int main()
{
        int n, sum,i;
        printf(“enter an integer “);
        scanf(“%d”,&n);
        sum=0;
        for(i=1;i<=n;i++)
        {
                sum =sum+i;
        }
        printf(“\n the sum of %d numbers is %d\n”, n,sum);
return 0;
}
```

/*program to find sum 1+1/2+1/3+1/4…….+1/n using for loop*/

```
#include<stdio.h>
int main()
{
        int n, i;
        float sum=0.0;
        printf(“enter an integer”);
        scanf(“%d”,&);
        for (i=1;i<=n;i++)
        {
                Sum=sum+1.0/i;
        }
```

```
            printf(“\n The sum of series is %f\n”,sum);
return 0;
}
```