

CHATER-1

Basics of C

1.1 INTRODUCTION TO NUMBER SYSTEM

A number system is a system of writing for expressing numbers. It is the mathematical notation for representing numbers of a given set by using digits or other symbols in a consistent manner. It provides a unique representation to every number and represents the arithmetic and algebraic structure of the figures. It also allows us to operate arithmetic operations like addition, subtraction, and division.

Different number systems are mentioned below.

1. Decimal number system (Base- 10)
2. Binary number system (Base- 2)
3. Octal number system (Base- 8)
4. Hexadecimal number system (Base- 16)

Computer numeral system

When we type any letter or word, the computer translates them into numbers since computers can understand only numbers. A computer can understand only a few symbols called digits, and these symbols describe different values depending on the position they hold in the number.

The value of any digit in a number can be determined by

- The digit
- Its position in the number
- The base of the number system

Decimal Number System

Decimal number system has base 10 because it uses ten digits from 0 to 9. In decimal number system, the positions successive to the left of the decimal point represent units, tens, hundreds, thousands and so on.

Every position shows a particular power of the base (10). For example, the **decimal number** 1457 consists of the digit 7

in the units position, 5 in the tens place, 4 in the hundreds position, and 1 in the thousands place whose value can be written as

$$(1 \times 1000) + (4 \times 100) + (5 \times 10) + (7 \times 1) = (1 \times 10^3) + (4 \times 10^2) + (5 \times 10^1) + (7 \times 1)$$

$$= 1000 + 400 + 50 + 7 = 1457$$

Base 2 Number System

Base 2 number systems are also known as Binary number system wherein, only two binary digits exist, i.e., 0 and 1. Specifically, the usual base-2 is a radix of 2. The figures described under this system are known as binary numbers which are the combination of 0 and 1. For example, 110101 is a binary number.

We can convert any system into binary and vice versa.

For Example, to write $(14)_{10}$ as binary number

Solution:

2	14	
2	7	0
2	3	1
1	1	1

Fig: 1.1

$$(14)_{10} = (1110)_2$$

Base 10 Number System

This system is expressed in decimal numbers. The base to the decimal is 10. This shows that there are ten symbols, 0 to 9. Similarly, the system using the symbols 0, 1, two will be of base 3, four symbols will be of base 4 and so on.

(1) Binary Number System

A Binary number system has only two digits that are **0 and 1**. Every number (value) represents with 0 and 1 in this number system. The base of binary number system is 2, because it has only two digits.

(2) Octal number system

Octal number system has only eight (8) digits from **0 to 7**. Every number (value) represents with 0, 1, 2, 3, 4, 5, 6 and 7

in this number system. The base of octal number system is 8, because it has only 8 digits.

(3) Decimal number system

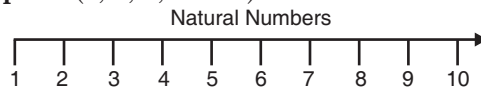
Decimal number system has only ten (10) digits from **0 to 9**. Every number (value) represents with 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 in this number system. The base of decimal number system is 10, because it has only 10 digits.

(4) Hexadecimal number system

A Hexadecimal number system has sixteen (16) alphanumeric values from **0 to 9** and **A to F**. Every number (value) represents with 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F in this number system. The base of hexadecimal number system is 16, because it has 16 alphanumeric values. Here **A is 10, B is 11, C is 12, D is 13, E is 14 and F is 15**.

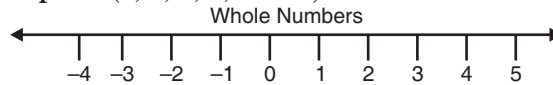
Natural Numbers – All positive or non-negative counting numbers. The set of natural numbers are commonly denoted as N.

Example – (1, 2, 3, 4..... ∞).



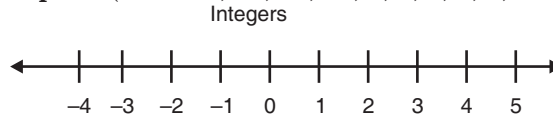
Whole Numbers – If we add zero in natural numbers set then it becomes whole numbers set.

Example – (0, 1, 2, 3, 4..... ∞).



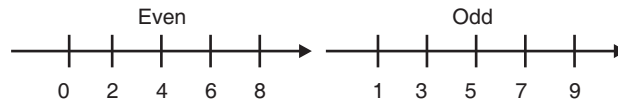
Integers – Integers are all whole numbers which include negative numbers as well as positive numbers.

Example – (∞-4, -3, -2, -1, 0, 1, 2, 3, 4, 5..... ∞).



Even & Odd Numbers – If the number is divided by 2 then it is called even number and if it is not then the numbers are called odd numbers.

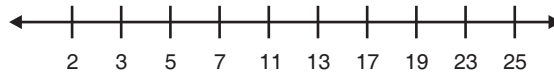
Even and Odd numbers



Example – (0, 2, 4, 6, 8, 10, 12..... ∞) are even numbers and (1, 3, 5, 7, 9, 11, 13, 15, 17, 19..... ∞) are odd numbers.

Prime Numbers – If a number is divided by itself only then it is called prime number. Prime Numbers can be positive or negative except 1.

Prime Numbers

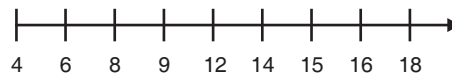


Example – (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61..... ∞)

Composite Numbers – Natural numbers which are not prime are called composite numbers.

Example – (4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, ∞)

Composite Numbers



1.2 INTRODUCTION TO FLOWCHART AND ALGORITHM

A **flowchart** is a type of diagram that represents an algorithm, work flow or process. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem.

Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

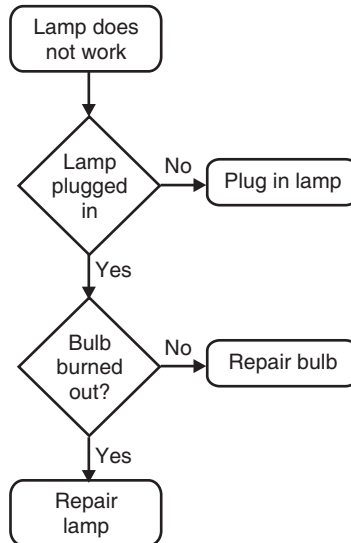

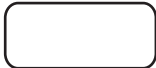

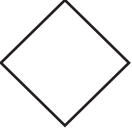

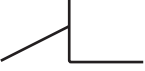


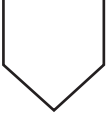


Fig. 1.2

A simple flowchart representing a process for dealing with a non-functioning lamp.

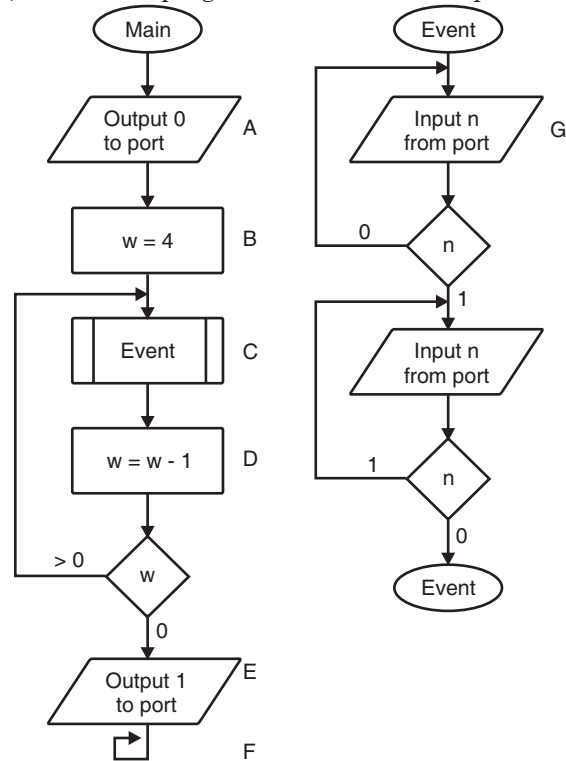
<i>ANSI/ISO</i>	<i>Shape</i>	<i>Name Description</i>
	Flow line (Arrow-head)[15]	Shows the process's order of operation. A line coming from one symbol and pointing at another.[14] Arrowheads are added if the flow is not the standard top-to-bottom left-to right.[15]
	Terminal[14]	Indicates the beginning and ending of a program or sub-process. Represented as a stadium [14] oval or rounded (fillet) rectangle. They usually contain the word "Start" or "End" or another phrase signaling the start or end of a process such as "submit inquiry" or "receive product".

	Process[15]	Represents a set of operations that changes value form or location of data. Represented as a rectangle. [15]
	Decision[15]	Shows a conditional operation that determines which one of the two paths the program will take.[14] The operation is commonly a yes/no question or true/false test. Represented as a diamond (rhombus).[15]
	Input/Output[15]	Indicates the process of inputting and outputting data [15] as in entering data or displaying results. Represented as a parallelogram. [14]
	Annotation[14] (Comment) [15]	Indicating additional information about a step the program. Represented as an open rectangle with a dashed or solid line connecting it to the corresponding symbol in the flowchart.[15]
	Predefined Process[14]	Shows named process which is defined elsewhere. Represented as a rectangle with double-struck vertical edges.[14]
	On-page Connector[14]	Pairs of labeled connectors replace long or confusing lines on a flowchart page. Represented by a small circle with a letter inside. [14][18]

	Off-page Connector[14]	A labeled connector for use when the target is on another page. Represented as a home plate shaped pentagon.[14][18]
---	------------------------	--

Example 1. The system has one input and one output. An event should be recognized when the input goes from 0 to 1 and back to 0 again. The output is initially 0, but should go 1 after four events are detected. After this point, the output should remain 1. Design a flowchart to solve this problem.

Solution: This example also illustrates the concept of a subroutine. We break a complex system into smaller components so that the system is easier to understand and easier to test. In particular, once we know how to detect an event, we will encapsulate that process into a subroutine, called Event. In this example, the main program first sets the output to zero, calls



the function Event four times, then it sets the output to one. To detect the 0 to 1 to 0 edges in the input, it first waits for 1, and then it waits for 0 again. The letters A through H in Figure 5.4 specify the software activities in this simple example. In this example, execution is sequential and predictable.

Algorithm - An algorithm is a set of instructions designed to perform a specific task. In computer programming, algorithms are often created as functions. These functions serve as small programs that can be referenced by a larger program. For example:-

1. **Step 1** is really just a reminder that this is a procedure with a beginning and an end.

2. **In step 2**, we make a place in the computer to store what the user types in, also called a variable

3. **In step 3**, we clear this variable because we might need to use it again and don't want the old contents mixed in with the new.

4. **In step 4**, we prompt the user for an email address

5. **In step 5**, we stick it in our nifty variable.

6. **In step 6**, we tell our computer to take a close look at this email address—is it really an email address?

1.3 HISTORY OF C, WHERE C STANDS

History of 'C'

The root of all modern languages is ALGOL, introduced in the early 1960s.

ALGOL was the first computer language to use a block structure. Although it never become popular in USA, it was widely used in Europe. ALGOL gave the concept of structured programming to the computer science community.

In 1967, Martin Richards developed a language called BCPL (Basic combined programming language). In 1970, Ken Thompson created a language using many features of

BCPL and called it simply B. B was used to create early versions of UNIX operating system at **Bell laboratories**.

'C' was evolved from ALGOL, BCPL and by Dennis M. Ritchie at the Bell laboratories in 1972. The language became more popular after publication of the book "The C programming language" by Brian W. Kernighan and Dennis M. Ritchie in 1978. The book was so popular that the language came to be

known as “K & RC”.

The technical committee approved a version of C in december 1989 which is now known as ANSI C. (American National Standards)

- 1960 - ALGOL (International Group)
- 1967 - BCPL (Martin Richards)
- 1970 - B (Ken Thompson)
- 1972 - Traditional C (Dennis Ritchie)
- 1978 - K&RC (Kernighan and Ritchie)
- 1989 - ANSI C (ANS committee)
- 1990 - ANSI / ISO C (ISO Committee)
- 1999 - C 99 (Standardization Committee)

WHERE C STANDS

Let us now see how C compares with other programming languages. All the programming languages can be divided into two categories:

- Problem oriented languages or High level languages:

These languages have been designed to give a better programming efficiency, i.e., faster program development. Examples of language falling in this category are FORTRAN, BASIC, and PASCAL.....

- Machine oriented languages or Low level languages:

These languages have been designed to give a better machine efficiency, i.e., faster program execution. Examples of languages falling in this category are Assembly language and Machine language.

C stands between these two categories. That’s why it is often called a Middle level language, since it was designed to have both: a relatively good programming efficiency (as compared to Machine oriented languages) and relatively good machine efficiency (as compared to Problem oriented languages).

C is a High-level Language

C is often called a high level language. This does not mean that C is less powerful, harder to use, or less developed than a high level language such as BASIC or Pascal, nor does it imply that C has the cumbersome nature of assembly language. Rather, C is thought of as middle – level language because it

combines the best elements of high level languages with the control and flexibility of assembly languages.

As high level language, C allows the manipulation of bits, bytes and addresses. Despite this fact, C code is also very portable. Portability means that it is easy to adapt software written for one type of computer or operation system to another type.

High Level Languages are: C, Ada, Modula-2, Pascal, COBOL, FORTRAN, and BASIC.

Middle Level Languages are: Forth and

Low Level Languages are: Macro-assembler machine language.

Language Translators

Computers can understand only machine language instructions. Therefore, the program written in any other language should be translated into machine language.

There are two types of translators.

1. Compiler
2. Interpreter

Difference between Compiler and Interpreter

A compiler checks the entire program at a time and if it is error free then it produces the machine language instruction.

The interpreter translates one statement at a time and if it is error free it produces the machine language instruction. Executing a program written in a high level language is a two-step process.

- The code (source program) should be compiled (*i.e.*, by either compiler or interpreter) to produce machine language instructions.
- Then the machine instructions are loaded into memory and it gets executed.

COMPILERS vs. INTERPRETERS

It is important to understand that a computer language defines the nature of a program and not the way that the program will be executed. There are two general methods by which a program can be executed. It can be compiled, or it can be interpreted. Although programs written in any computer language can be compiled or interpreted, some languages are designed more for one form or execution than the other. For example, Java was

designed to be interpreted, and C was designed to be compiled. However, in the case of C, it is important to understand that it was specifically optimized as a compiled language.

In its simplest form, an interpreter reads the source code of your program one line at a time, performing the specific instructions contained in that line. This is the way earlier version of BASIC worked. In languages such as Java, a program's source code is first converted into an intermediary form that is then interpreted. In either case, a run-time interpreter is still required to be present to execute the program.

A compiler reads the entire program and converts it into object code, which is a translation of the program's source code into a form that the computer can execute directly. Object code is also referred to as binary code or machine code. Once the program is compiled, a line of source code is no longer meaningful in the execution of your program.

In general, an interpreted program runs, slower than a compiled program. Hence a compiler converts a program's source code into object code that a computer can execute directly.

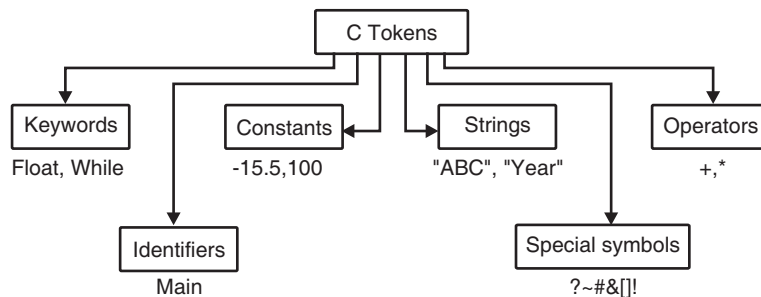
Therefore, compilation is a one-time cost, while interpretation incurs an overhead each time a program is run.

1.4 C CHARACTER SET, TOKENS, CONSTANTS, VARIABLES, KEYWORDS

Character Set : The characters in C are grouped into the following categories:

1. Letters
2. Digits
3. Special characters
4. White Spaces

C Tokens: The individual words and punctuation marks are called tokens. C has six types of tokens.



Constants: C refers to fixed values that do not change during the execution of a program.

C supports several types of Constants.

- Numeric constants – Integer and real constants
- Character constants – single character & string constants

Variable: - A variable is a data name that may be used to store a data value. *e.g.* : Total, Average etc.

Keywords and Identifiers: Every C word is classified as either a keyword or an identifier. All keywords have fixed meanings and these meanings cannot be changed. All keywords must be written in lower case.

Like: int, break, else, long, switch, case, char, float, for, void, goto, do, if, while.

1.5. C OPERATORS

(Arithmetic, Logical, assignment, relational, increment and decrement, conditional, bit wise, special, operator precedence), C expressions data types.

C supports a rich set of built in operators.

C operators can be classified into a number of categories. They include:-

(1) Arithmetic operators: - C provides all the basic arithmetic.

<i>Operator</i>	<i>Meaning</i>
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

(2) Relational operators: - C supports six relational operators in all.

<i>Operator</i>	<i>Meaning</i>
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
= =	is equal to
!=	is not equal to

(3) Logical operator: - C has the following three logical operators.

&& meaning logical AND
 || meaning logical OR
 ! meaning logical NOT

TRUTH TABLE

Table-1.1

<i>Op-1</i>	<i>Op-2</i>	<i>Value of the</i>	<i>expression</i>
		Op-1&& op-2	Op-1 op-2
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

(4) Assignment operators: - They are used to assign the result of an expression to a variable. Symbol ‘ = ’

V% = exp;

e.g. V = 5 or V = V + 1

(5) Increment and decrement operators: C allows two very useful operators not generally found in other languages.

[++ and --]

Like ++ x or x ++

(6) Conditional operator : A binary operator pair “ ? : “ is available in C to construct conditional expression of the form

exp1? exp2 : exp3

e.g. a=10; b= 15

x= (a>b)? a : b;

Sol. b *i.e.*, 15

(7) Bitwise operator: - C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. Bitwise operators may not be applied to float or double.

<i>Operators</i>	<i>Meaning</i>
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

(8) Special Operators: C Supports some special operators of interest such as comma operators, size of operators, pointer operators (. and ->). The comma and size of operators.

Data types: C language is rich in its data type. C supports three classes of data types:

- Primary data type
- Derived data type
- User-defined data type

All C compilers support five fundamental data types, namely

- Integer- int -% d
V%= exp;
- Character – Char-% c
- Floating point- float-% f
- Double-precision floating point- %lf
- Long integer – long int -%ld

1.6 FORMATTED INPUT, FORMATTED OUTPUT

Formatted output : The use of print of function for printing captions and numerical result. The general form of printf statement is:-

```
Printf ("Control String", arg1, arg2... argn);
```

e.g.:

```
Printf ("Programming in C");
Printf ("%d", x);
```

Formatted input : It refers to an input data that has been arranged in a particular format. The general form of Scanf is

scanf ("Control String", arg1, arg2... argn);

e.g.,
scanf ("%d", & x);

QUESTION

1. Convert hexadecimal value 16 to decimal.

A. 22_{10} B. 16_{10} C. 10_{10} D. 20_{10}

$$\begin{aligned} 16 &= 6*16^0 + 1*16^1 \\ &= 6+16 \\ &= 22(\text{Ans}) \end{aligned}$$

Printf ("Control String", arg1, arg2... argn);

scanf ("Control String", arg1, arg2... argn);

2. Convert the following decimal number to 8-bit binary. 187

A. 101110112 B. 110111012
C. 101111012 D. 101111002

$$2 \mid 187 \mid 1$$

$$2 \mid 93 \mid 1$$

$$2 \mid 46 \mid 0$$

$$2 \mid 23 \mid 1$$

$$2 \mid 11 \mid 1$$

$$2 \mid 5 \mid 1$$

$$2 \mid 2 \mid 0$$

$$\mid 1 \mid 1$$

Answer = 10111011.

3. Convert binary 11111110010 to hexadecimal.

A. $EE2_{16}$ B. $FF2_{16}$
C. $2FE16$ D. $FD2_{16}$

11111110010
 A=10, B =11, C=12, D=13, E=14, F=15.
 Select 4-4 pair group
 1111 1111 0010
 8421 8421 8421
 15 15 2
 F F 2
 Ans:(B).(FF2)₁₆.

4. Convert the following binary number to decimal. (01011)₂

- A. 11 B. 35
 C. 15 D. 10

$$(1*2^0)+(1*2^1)+(0*2^2)+(1*2^3)+(0*2^4)$$

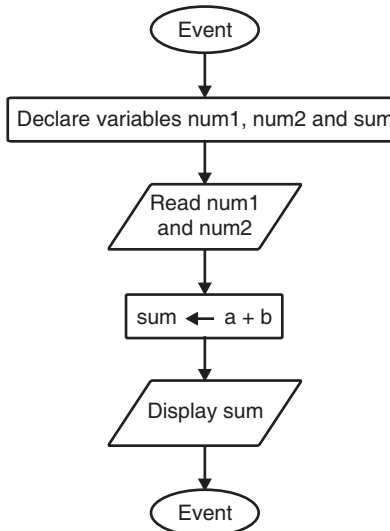
$$=1+2+0+8$$

$$=11(\text{Ans})$$

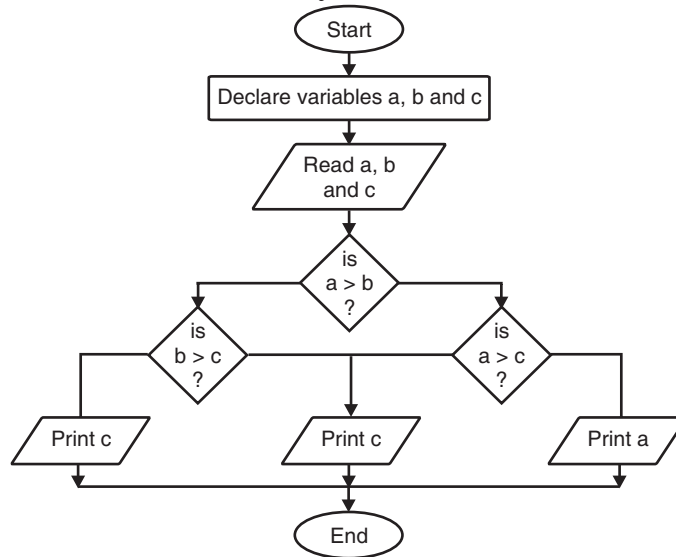
5. Convert the binary number 1001.0010₂ to decimal.

- A. 90.125 B. 9.125
 C. 125 D. 12.5

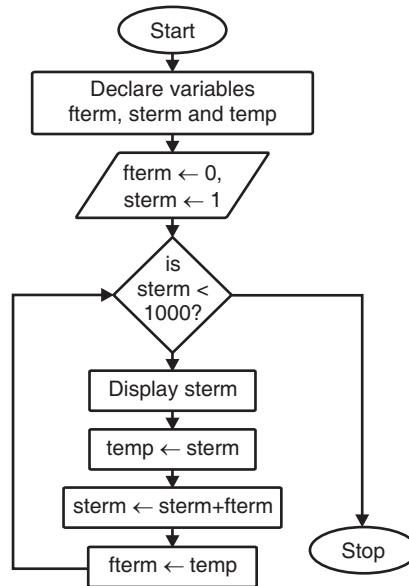
Draw a flowchart to add two numbers entered by user.

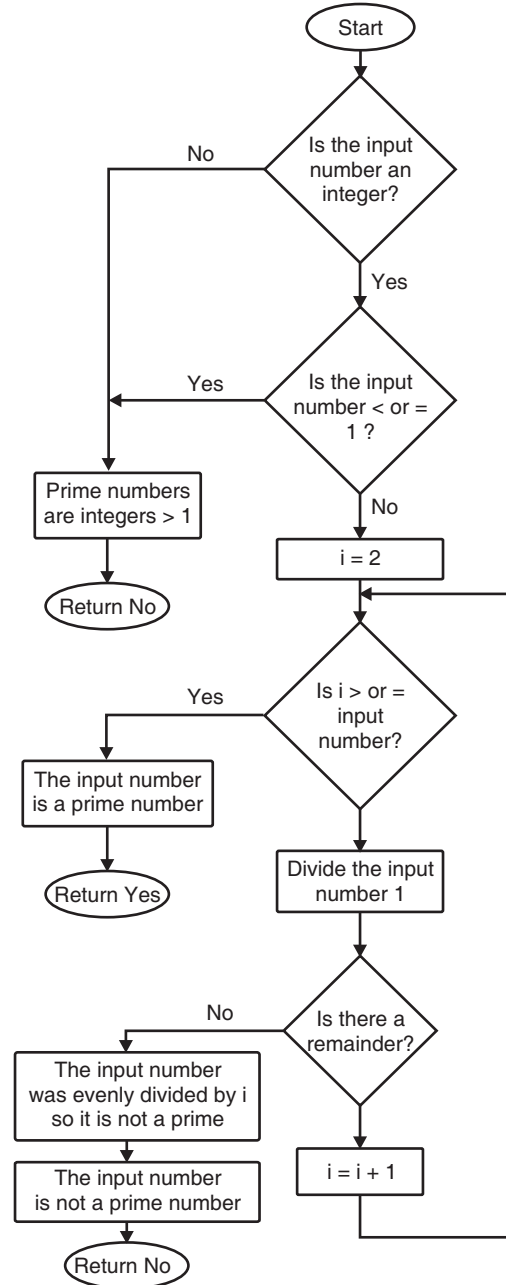


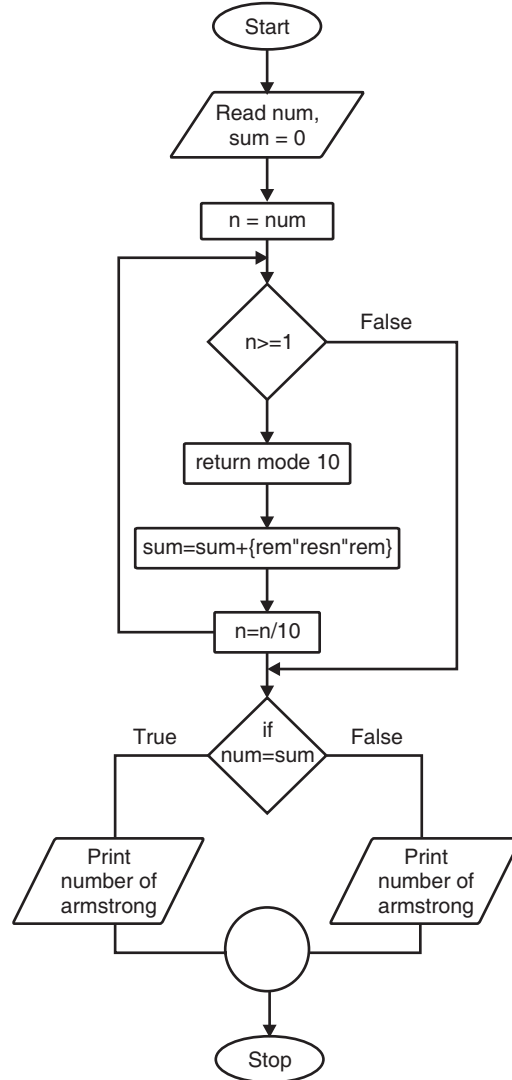
Draw flowchart to find the largest among three different numbers entered by user.



Draw a flowchart to find the Fibonacci series till term ≤ 1000 .



Finding Prime Numbers Flow Chart -

Flowchart for finding Armstrong number C program

Problem 1: An algorithm to calculate even numbers between 0 and 99

- Step 1. Start
- Step 2. $I \leftarrow 0$
- Step 3. Write I in standard output
- Step 4. $I \leftarrow I + 2$
- Step 5. If ($I \leq 98$) then go to line 3
- Step 6. End

Problem 2: Design an algorithm which gets a natural value, n, as its input and calculates odd numbers equal or less than n. Then write them in the standard output

- Step 1. Start
- Step 2. Read n
- Step 3. $I \leftarrow 1$
- Step 4. Write I
- Step 5. $I \leftarrow I + 2$
- Step 6. If ($I \leq n$) then go to line 4
- Step 7. End

Problem3: Design an algorithm which generates even numbers between 1000 and 2000 and then prints them in the standard output. It should also print total sum:

- Step 1. Start
- Step 2. $I \leftarrow 1000$ and $S \leftarrow 0$
- Step 3. Write I
- Step 4. $S \leftarrow S + I$
- Step 5. $I \leftarrow I + 2$
- Step 6. If ($I \leq 2000$) then go to line 3 else go to line 7
- Step 7. Write S
- Step 8. End

Problem4: Design an algorithm with a natural number, n, as its input which calculates the following formula and writes the result in the standard output: $S = \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{n}$

- Step 1. Start
- Step 2. Read n
- Step 3. $I \leftarrow 2$ and $S \leftarrow 0$

Step 4. $S = S + 1/I$
 Step 5. $I \leftarrow I + 2$
 Step 6. If $(I \leq n)$ then go to line 4 else write S in standard
 output
 Step 7. End

Problem 5: A algorithm to find the largest value of any three numbers.

Step1: Start
 Step2: Read/input A, B and C
 Step3: If $(A \geq B)$ and $(A \geq C)$ then Max=A
 Step4: If $(B \geq A)$ and $(B \geq C)$ then Max=B
 Step5: If $(C \geq A)$ and $(C \geq B)$ then Max=C
 Step6: Print Max
 Step7: End

Qus.1 write a C programming for display "Hello World"

```
#include <stdio.h>
int main()
{
    // printf() displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
```

Qus. 2. write a C programming for Print the integer.

```
#include<stdio.h>
int main()
{
    int number;

    // printf() displays the formatted output
    printf("Enter an integer: ");

    // scanf() reads the formatted input and stores them
    scanf("%d", &number);
```

```
// printf() displays the formatted output
printf("You entered: %d", number);
return 0;
}
```

Qus. 3 write a C programming for add two no.

```
#include<stdio.h>
int main()
{
int first number, secondNumber, sumOfTwoNumbers;
printf("Enter two integers: ");

// Two integers entered by user is stored using scanf()
function
scanf("%d %d", &firstNumber, &secondNumber);

// sum of two numbers in stored in variable sumOfTwoNum-
bers
sumOfTwoNumbers=firstNumber+secondNumber;

// Displays sum
printf("%d + %d = %d", firstNumber, secondNumber, su-
mOfTwoNumbers);
return 0;
}
```

Qus. 4. Write a C programming for Multiplication two no.

```
#include<stdio.h>
int main()
{
int first number, secondNumber, multOfTwoNumbers;

printf("Enter two integers: ");

// Two integers entered by user is stored using scanf()
function
scanf("%d %d", &firstNumber, &secondNumber);
```

```
// sum of two numbers in stored in variable sumOfTwoNum-
bers
multOfTwoNumbers=firstNumber*secondNumber;

// Displays sum
printf("%d x %d = %d", firstNumber, secondNumber, mul-
tOfTwoNumbers);
return 0;
}
```

REVIEW EXERCISE

- Q.1. Who invented C Language?
- Q.2. Define what is C Token?
- Q.3. Define what is a Keyword?
- Q.4. Define what is data types?
- Q.5. Explain the types of operators.
- Q.6. Write a C program to display "Welcome to C".