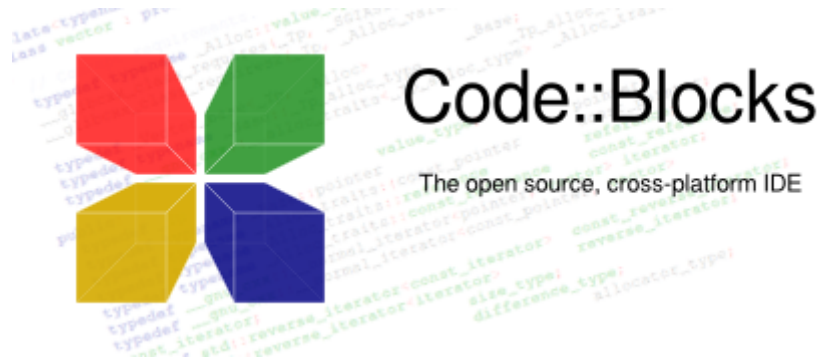


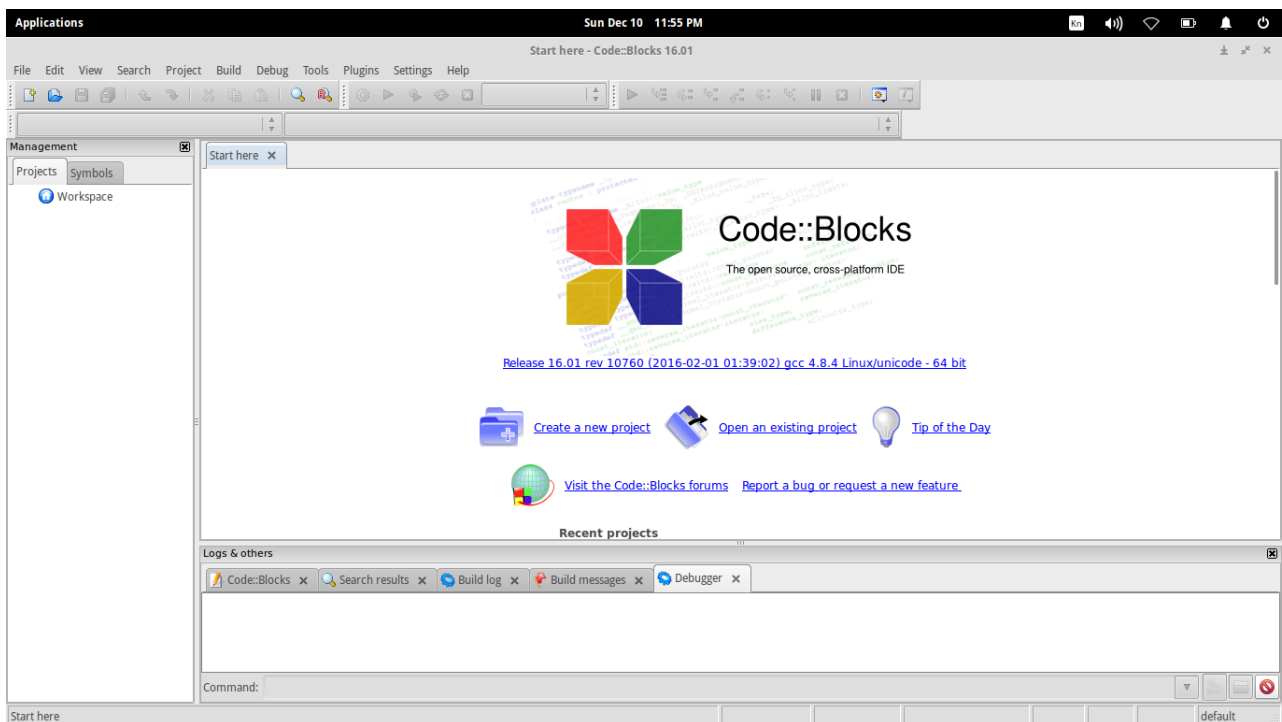
# CODE::BLOCKS



## CODE::BLOCKS TUTORIAL

### Using Code::Blocks to compile and execute a C Program

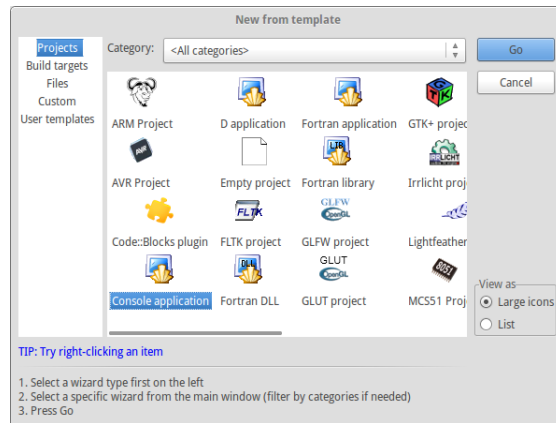
To launch Code::Blocks IDE, Click on  
**Applications** → **Programming** → **Code::Blocks IDE**  
You will get a window as shown below



Code::Blocks creates what is called a Workspace to keep track of the project you are working on. It is possible for you to be working on multiple projects within your workspace. A project is a collection of one or more source (as well as header) files. Source files are the files that contain the source code for your program. If you are developing a C program, you are writing C source code (.c files).

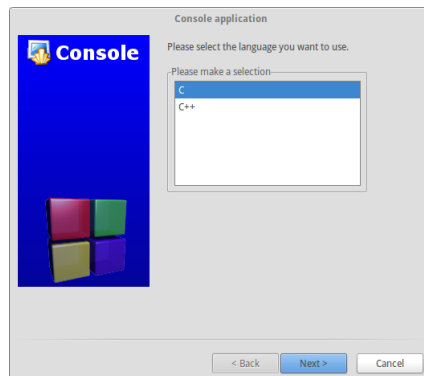
First start a new Project by clicking on **Create a new project**

**OR** To create a project, click on the **File** pull-down menu, open **New** and then **Project**. You get this pop-up window



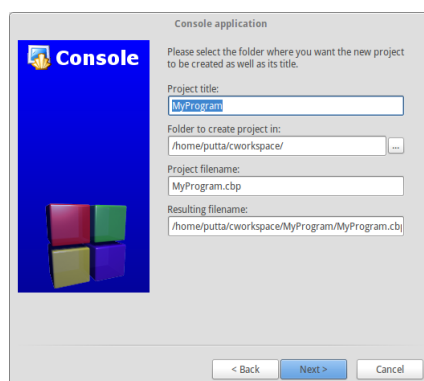
Choose Console application

Then in the next window select the programming language C (and not C++)

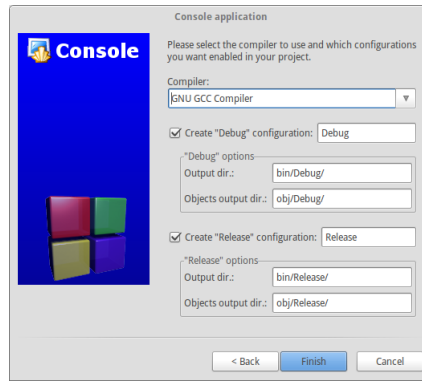


In the next step give the Project Title and specify the folder where you want to save your project.

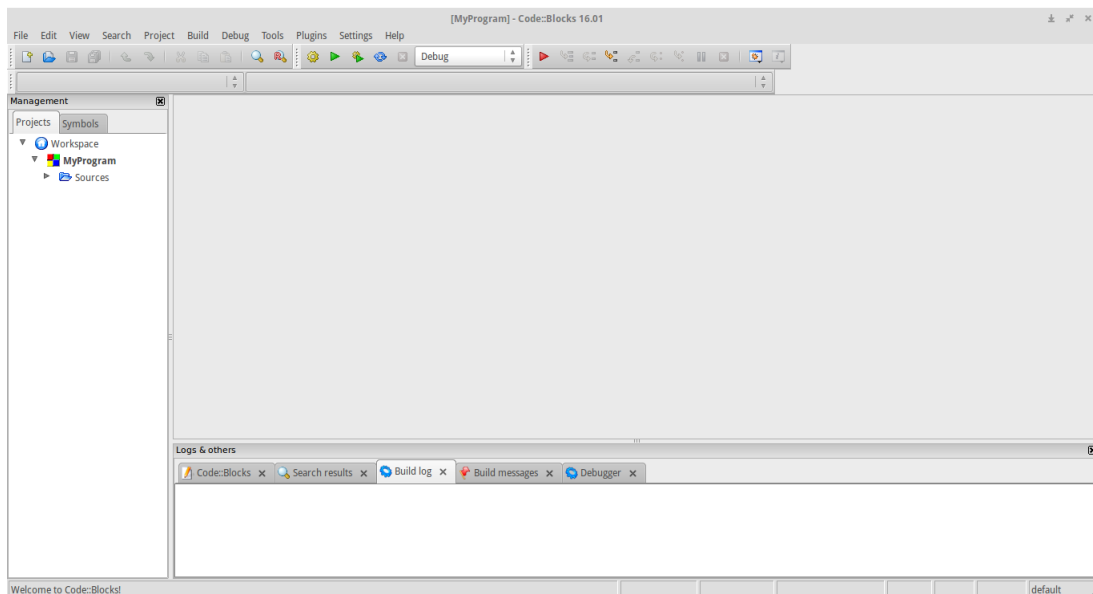
(**Note :** Dont use any special characters or whitespaces for project title and folder names)



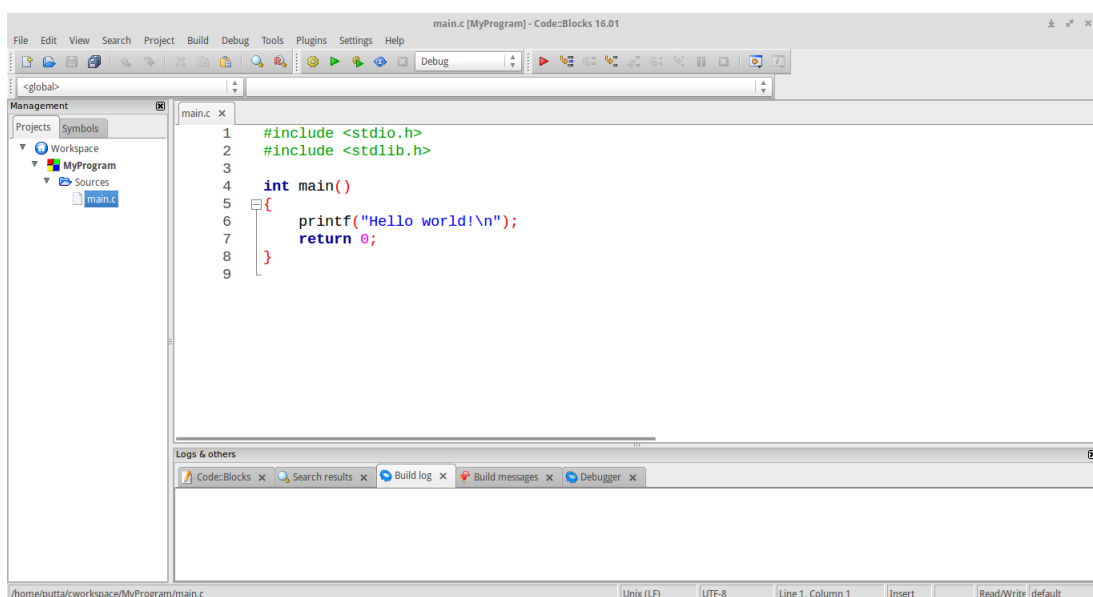
Finally you will be prompted to choose the compiler. Just choose the default options here (Dont change the options). You should be using GNU GCC Compiler. Click Finish to create the new project.



The system will then return to the [MyProgram] window and you are ready to write your program. In the Management area of the screen (Shift-F2 toggles the Management display), you will see the files that are part of the project in the Projects tab. To see the source files, click on the triangle symbol to expand the Workspace and its subdirectories.



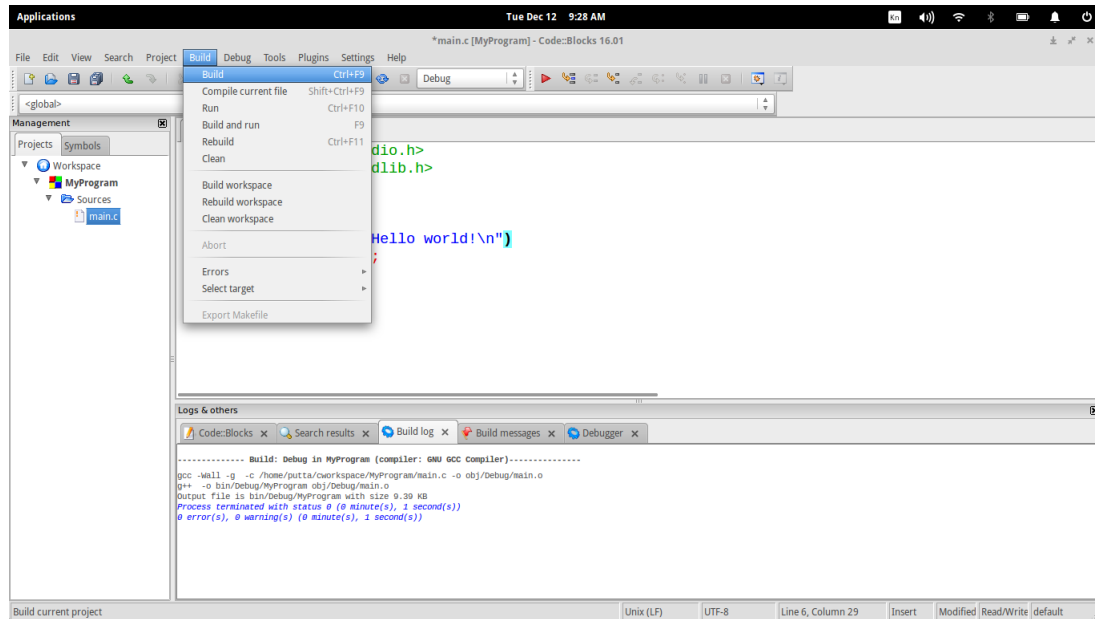
Under Sources, there is a file called **main.c**, which is automatically created for you when you build a console application. Click on **main.c**.



main.c contains a simple Hello World program which you can edit later to solve a programming problem. Now let us see how to compile and execute this main.c program. Just to understand the process of debugging we knowingly introduce an error in the program by removing the semicolon after the printf statement. We will now compile the program (To compile a file means to take the instructions that you have written and translate it into machine code

for the computer to understand).

Compile the project from the Build pull-down menu by clicking on **Build** option[**Ctrl+F9**].



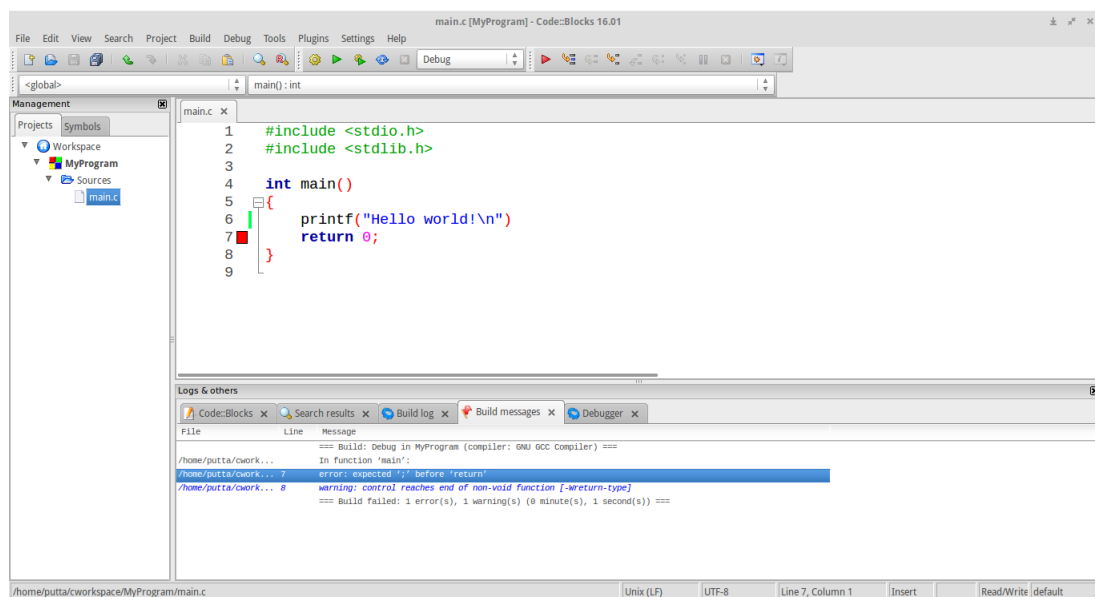
The error messages are shown in the Build messages window at the bottom. Let us now try to understand these error messages. === Build: Debug in MyProgram (compiler: GNU GCC Compiler) ===

main.c In function main:

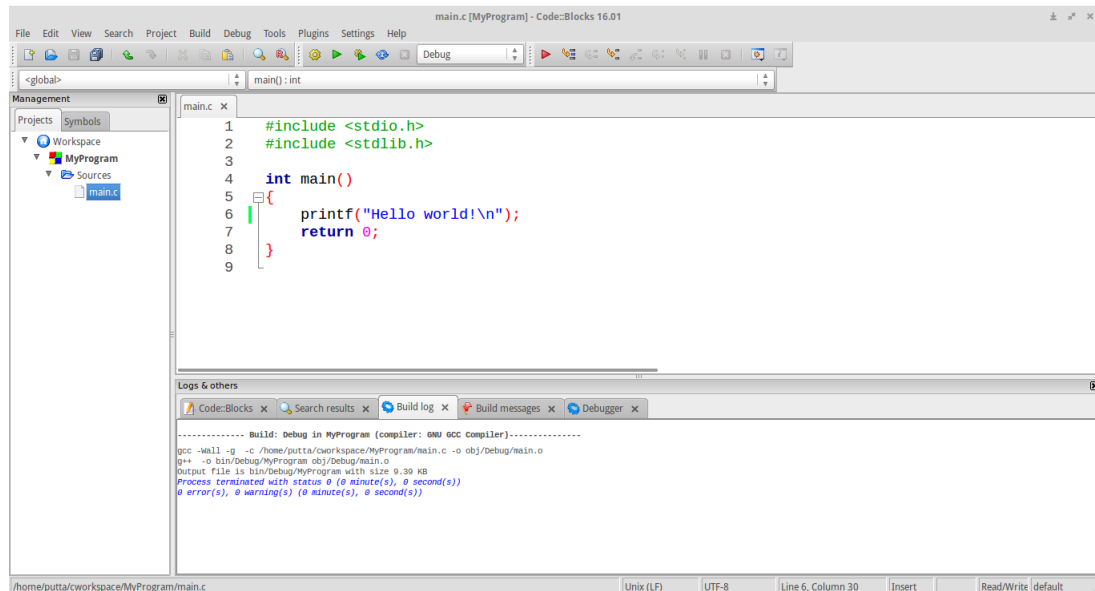
main.c 7 error: expected ; before return

main.c 8 warning: control reaches end of non-void function [-Wreturn-type]

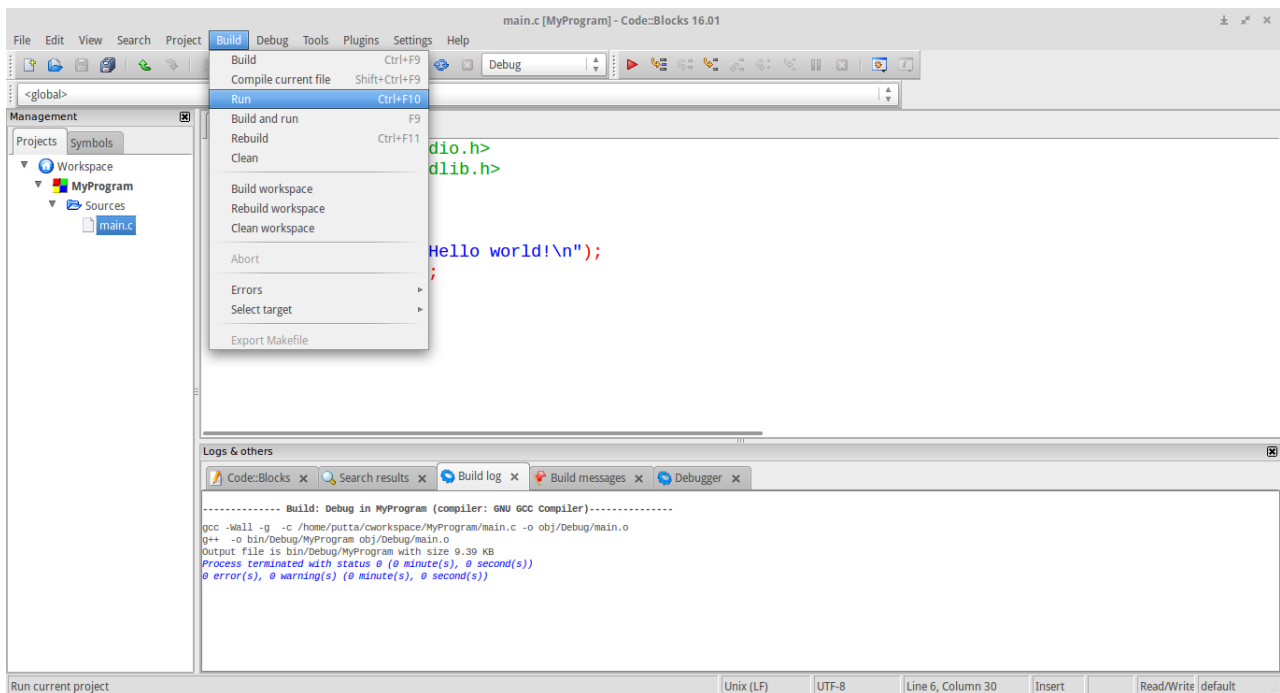
=== Build failed: 1 error(s), 1 warning(s) (0 minute(s), 1 second(s)) ===



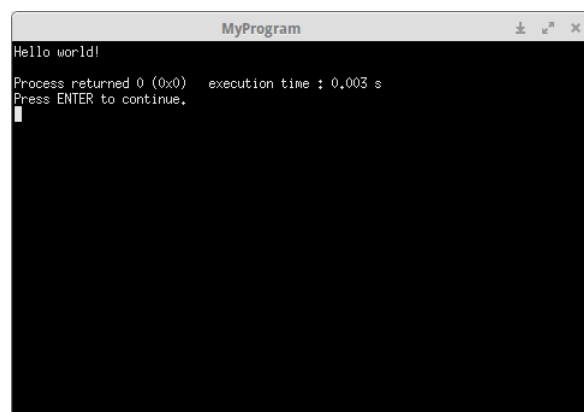
The error messages show the errors in syntax (and not the logical errors). It also indicates the line number and the type of error. Here in this example the error says that before the return statement in line no 7 a semicolon is missing. The next message is a warning message which has resulted because of the previous error. Now go to line number 6 and add a semicolon at the end. Now build your project again.



Now the build message window shows the following message. 0 error(s), 0 warning(s) (0 minute(s), 0 second(s)) This means that the errors and warnings have been successfully resolved. Now it is time to run the program. You can Execute the project from the Build pull-down menu by clicking on **Run** option[Ctrl+F10].



An output window pops displaying the output of the program. A greeting message "Hello world!" is printed on to the output console.



## Installing Code::Blocks IDE

If you want to install Code::Blocks IDE on your system, you can download it from its official webpage.

<http://www.codeblocks.org/>

It is available for Linux, Mac, Windows platforms.

On Debian/Ubuntu systems Code::Blocks can be installed by using the **apt** package manner. You have to be connected to the internet for necessary files to be downloaded. Run the following command.

**\$sudo apt install codeblocks**

For Windows download the binary whose name is like **codeblocks-17.12mingw-setup.exe**

---

## Algorithm

- An algorithm is a basic tool which is used to express the solution for a given problem systematically in the form of instructions. This solution is called logic. It takes a set of input values and produces the desired output.
- An algorithm is defined as unambiguous, step by step procedure (instructions) to solve a given problem in finite number of steps by accepting a set of inputs and producing the desired output. After producing the result, the algorithm should terminate. Usually, Algorithms are written in simple English like statements along with simple mathematical expressions.

### Characteristics of an Algorithm:

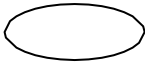

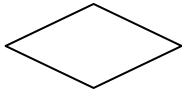



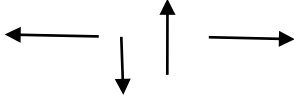
- **Input:** It may accept a zero or more inputs.
- **Output:** It should produce at least one output (result).
- **Definiteness:** Each instruction must be clear, well defined and precise. There should not be any ambiguity.
- **Finiteness:** It should be a sequence of finite instructions. That is, it should end after a fixed time. It should not enter into an infinite loop.
- **Effectiveness:** This means that operations must be simple and are carried out in a finite time at one or more levels of complexity. It should be effective whenever traced manually for the results.

### Algorithmic Notations:

- Name of the Algorithm
  - Step number
  - Explanatory Comment
  - Termination
-

## Flowchart

- A flow chart is a pictorial representation of an algorithm. That is flowchart consists of sequence of instructions that are carried out in an algorithm. All the steps are drawn form of different shapes of boxes, circle and connecting arrows. Flowcharts are mainly used to help programmer to understand the logic of the program.
- The various types of geometric shapes, arrows and symbols used while drawing the flowchart are called flowchart symbols. The symbols used and the meaning associated with each symbol are shown below.

<b><u>Symbols used</u></b>	<b><u>Meaning associated with symbols</u></b>
	<b>Start or end of the flowchart (program)</b>
	<b>Input or output operation</b>
	<b>Decision making and branching</b>
	<b>Connector</b>
	<b>Predefined computation or process (used with functions are used)</b>
	<b>Repetition or a loop which is normally used to execute a group of instructions for a specifies number of times</b>
	<b>Flow of control</b>



---

**1. Develop a Program to solve simple computational problems using arithmetic expressions and use of each operator leading to simulation of a Commercial calculator (no Built in math function).**

**Algorithm**

Step 1: Start

Step 2: Read the values of a and b.

Step 3: Read Ch

Step4: Verify ch value by using switch statement if ch=1 then goto step 5 or if ch=2 then goto step 6 or if read ch is 3 then goto step 7 or if read ch is 4 then goto step 8 or if read ch is 5 then goto step 9 or otherwise goto step 10.

Step 5: if ch=1 then compute  $res=a+b$  and goto step 11.

Step 6: if ch=2 then compute  $res=a-b$  and goto step 11.

Step 7: if ch=3 then compute  $res=a*b$  and goto step 11.

Step 8: if ch=4 then compute  $res=a/b$  and goto step 11.

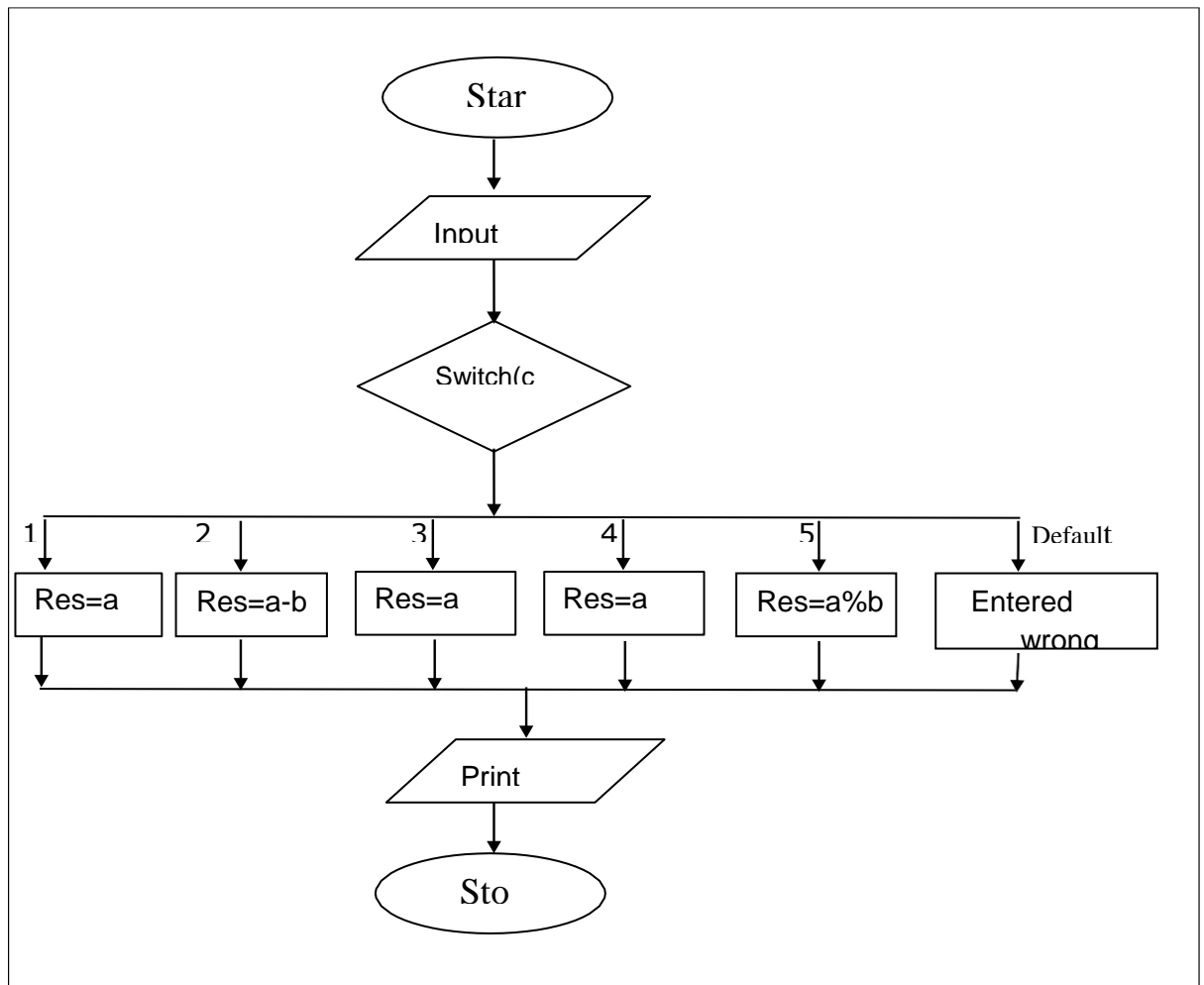
Step 9: if ch=5 then compute  $res=a\%b$  and goto step 11.

Step 10: if ch value is other than above options then print wrong choice entered then goto step 12.

Step 11: Print res and goto step 12

Step 12: stop.

## Flowchart



---

## **Program**

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int ch,a,b;
    float res;
    printf("Enter two numbers:\n");
    scanf("%d%d",&a,&b);
    printf("1=Add, 2=Sub, 3=Mul, 4=Div, 5=Rem\n");
    printf("Enter your choice:\n");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1: res=a+b;
                break;
        case 2: res=a-b;
                break;
        case 3: res=a*b;
                break;
        case 4: res=(float)a/b;
                break;
        case 5: res=a%b;
                break;
        default: printf("Entered Wrong choice\n");
    }
    printf("Result=%f\n",res);
}
```

---

2. Develop a program to compute the roots of a quadratic equation by accepting the coefficients. Print the appropriate messages.

**Algorithm**

Step 1: Start

Step 2: Read the values of non-zero coefficients a,b and c.

Step 3: If a is zero goto Step 9. Otherwise Compute the value of discriminant (disc) which is Equal to  $(b*b)-(4*a*c)$ .

Step 4: Check if disc is equal to 0. If true, then go to Step 5. Otherwise, goto Step 6

Step 5: Compute the roots.

$$\text{root1} = (-b)/(2*a)$$

$$\text{root2} = \text{root1}$$

Output the values of roots, root1 and root2. Go to Step 9

Step 6: Check if disc is greater than zero or not. If true, then go to Step 7. Otherwise, goto Step 8.

Step 7: Compute the real and distinct roots,

$$\text{root1} = (-b + \sqrt{\text{disc}})/(2*a)$$

$$\text{root2} = (-b - \sqrt{\text{disc}})/(2*a)$$

Output the values of roots, root1 and root2. Go to Step 9.

Step 8: Compute the complex and distinct roots.

$$\text{Compute the real part, } r\_part = (-b)/(2*a)$$

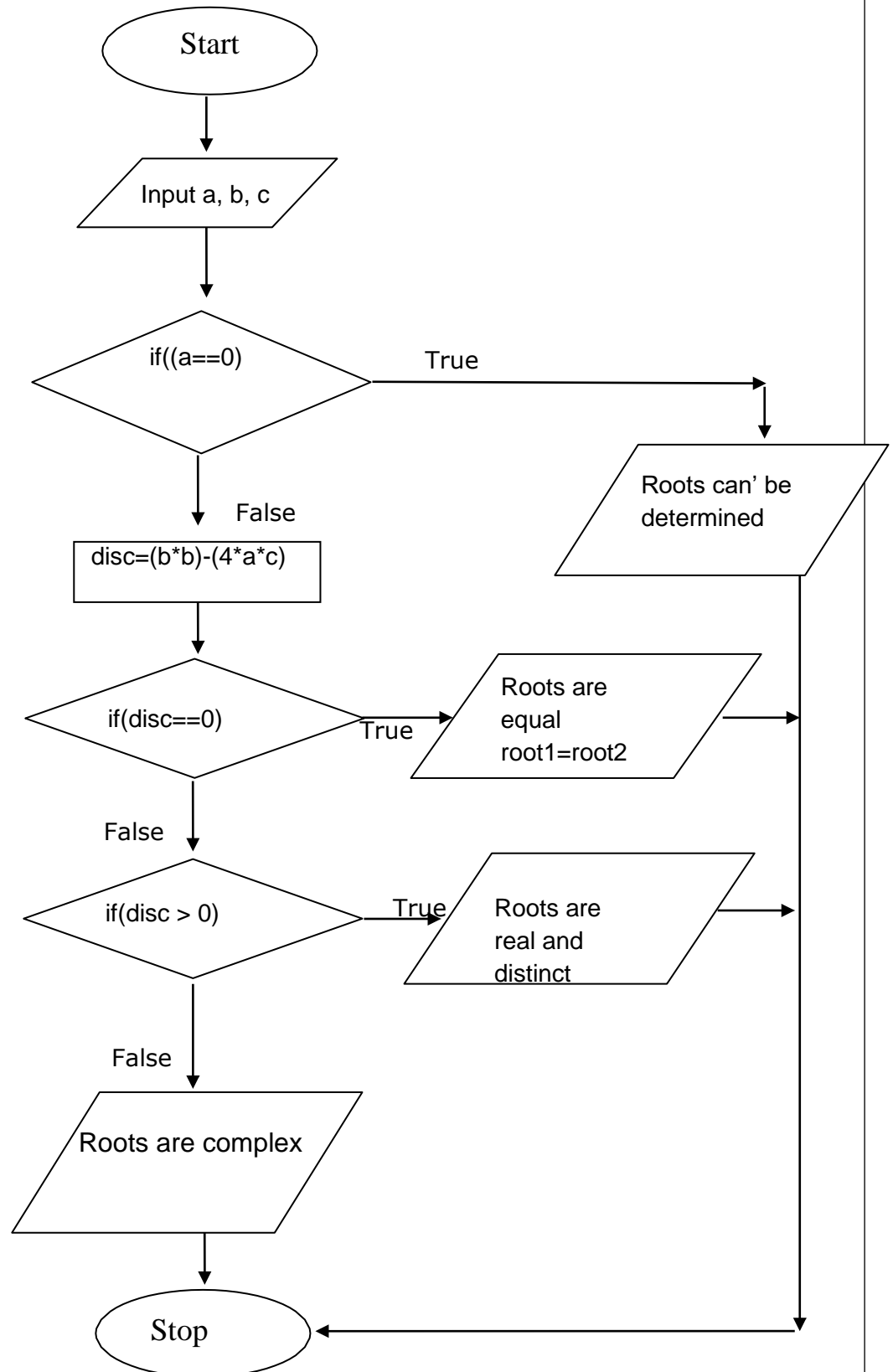
$$\text{Compute the imaginary part, } i\_part = \sqrt{-\text{disc}}/(2*a)$$

Output roots as

$$\text{root1} = r\_part + i\_part$$

$$\text{root2} = r\_part - i\_part$$

Step 9: Stop.

**Flow chart**

---

## **Program**

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main( )
{
    float a, b, c, root1, root2, rpart, ipart, disc;
    printf("Enter the 3 coefficients:\n");
    scanf("%f%f%f", &a, &b, &c);
    if(a == 0)
    {
        printf("Roots cannot be Determined:\n");
        exit(0);
    }
    disc = (b*b) - (4*a*c);
    if(disc == 0)
    {
        printf("Roots are equal\n");
        root1=root2= -b / (2*a);
        printf("root1=root2=%f",root1);
    }
    else if(disc>0)
    {
        printf("Roots are real and distinct\n");
        root1= (-b + sqrt(disc)) / (2*a);
        root2= (-b - sqrt(disc)) / (2*a);
        printf ("root1 = %f \n root2 = %f\n", root1, root2);
    }
    else
    {
        printf("Roots are complex\n");
        rpart = -b / (2*a);
        ipart = (sqrt (-disc)) / (2*a);
        printf("root1 = %f + i %f \n", rpart, ipart);
        printf("root2 = %f - i %f \n", rpart, ipart);
    }
}
```

Output

**3. An electricity board charges the following rates for the use of electricity: for the first 200 units 80 paise per unit: for the next 100 units 90 paise per unit: beyond 300 units rupees 1 per unit. All users are charged a minimum of rupees 100 as meter charge. If the total amount is more than Rs 400, then an additional Surcharge of 15% of total amount is charged. Write a program to read the name of the user, number of units consumed and print out the charges.**

**Algorithm:**

Input: Customer name and unit consumed

Output: Customer name, units consumed and total amount to be paid

**Step 1:** Start

**Step 2:** Read the name of customer and the unit consumed by the customer.

**Step 3:** Check if the unit consumed is greater than 1 and less than 200, if true goto step 4 else goto step 5.

**Step 4:** Compute:  $\text{amt} = 100 + (0.8 * \text{units})$ .

**Step 5:** if unit is greater than 200 and less than 300, if true goto step 6 else goto step 7

**Step 6:** Compute  $\text{amt} = 100 + (200 * 0.8) + ((\text{units} - 200) * 0.9)$

**Step 7:** Compute  $\text{amt} = 100 + (200 * 0.8) + (100 * 0.9) + ((\text{units} - 300) * 1)$ , then goto step 8

**Step 8:** Check if the amt is less than or equal to 400, if true goto step 9 otherwise goto step 10.

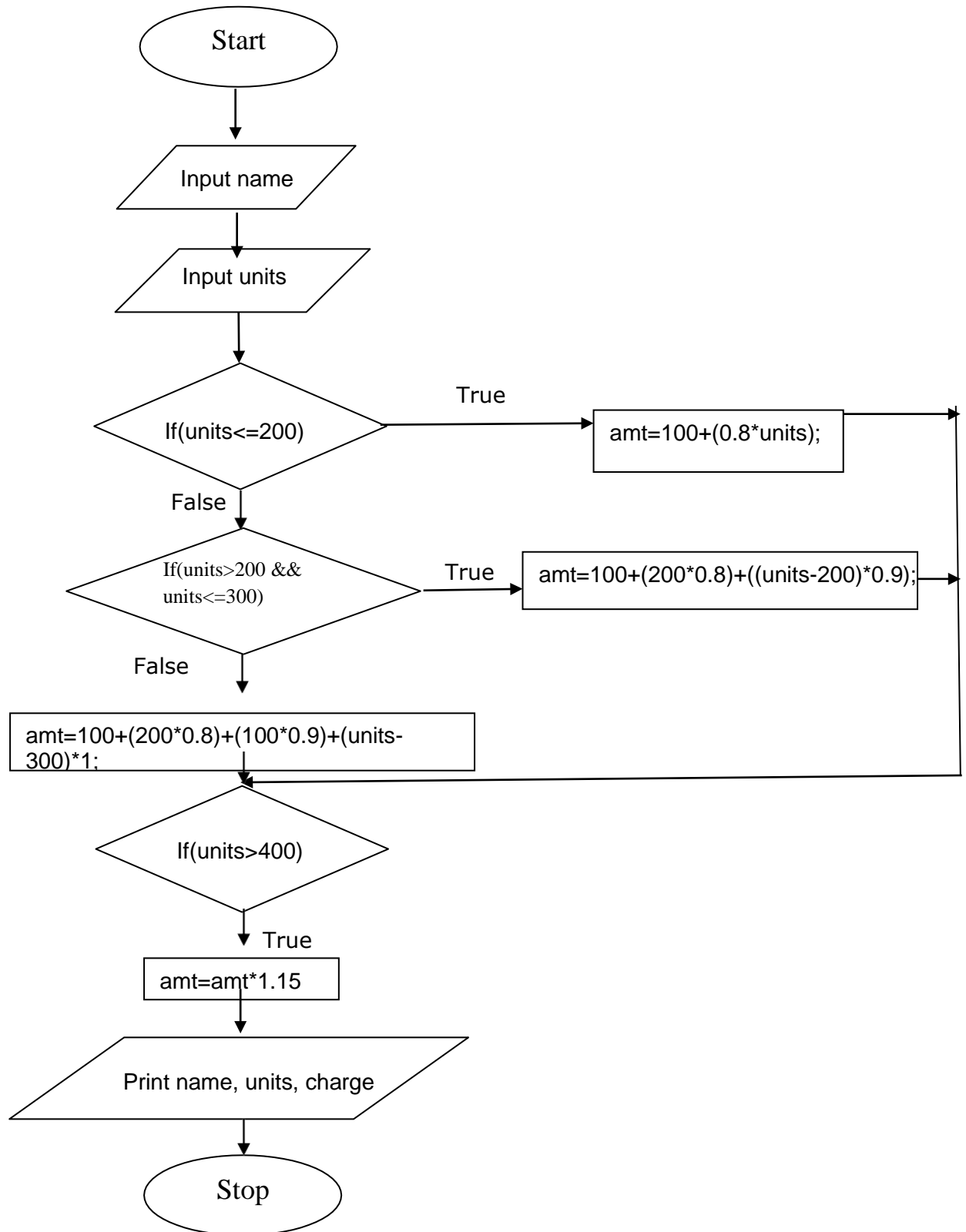
**Step 9:** Print the amount charged and goto step 11.

**Step 10:** compute  $\text{amt} = \text{amt} * 1.15$ , and print the amount charged.

**Step 11:** Stop .



## Flow chart



**Program**

```
#include <stdio.h>

void main()
{
    char name[10];
    float unit, amt;
    printf("Enter your name and unit Consumed:");
    scanf("%s %f",name,&unit);
    if(unit<=200)
        amt=unit*0.80+100;
    else if((unit>200)&&(unit<=300))
        amt=200*0.80+((unit-200)*0.90)+100;
    else
        amt=200*0.80+100*0.90+((unit-300)*1)+100;
    if(amt>400)
        amt=1.15*amt;
    printf("Name: %s\n Unit=%f \n charge=%f ",name,unit,amt);
}
```

Output

---

---

4. Write a C Program to display the following by reading the number of rows as input,

1  
1 2 1  
1 2 3 2 1  
1 2 3 4 3 2 1  
-----  
nth row

#### Algorithm

Input: Number of rows

Output: print the rows

**Step 1:** Start

**Step 2:** Read the number of rows by the user.

**Step 3:** print blank spaces.

**Step 4:** Display number in ascending order upto middle

**Step 5:** Display number in reverse order after middle

**Step 6:** print each row in new line and go to step 4

**Step 7:** print the number of rows given as input

**Step 8:** Stop.

---

## **Flowchart**



---

## **Program**

```
#include <stdio.h>

void main()
{
    int i,j,n;
    printf("Input number of rows : ");
    scanf("%d",&n);
    for(i=0;i<=n;i++)
    {
        /* print blank spaces */
        for(j=1;j<=n-i;j++)
            printf(" ");
        /* Display number in ascending order upto middle*/
        for(j=1;j<=i;j++)
            printf("%d",j);

        /* Display number in reverse order after middle */
        for(j=i-1;j>=1;j--)
            printf("%d",j);

        printf ("\n");
    }
}
```

## **Output:**

---

---

**5. Introduce 1D Array manipulation and implement binary search.**

**Algorithm**

**Step 1:** Start

**Step 2:** Read size of the array  $n$

**Step 3:** Read the list of elements in sorted order  $a[i]$ .

**Step 4:** Read the key element in key

**Step 5:** initialize **low=0** and **high= n-1**

**Step 6:** Check if low is less than or equal to high. If condition is true, goto step 7, other wise goto Step 11

**Step 7:** find the middle element.i.e.  $mid=(low+high)/2$ ;

**Step 8:** if key element is equal to mid element, then initialize loc value, **loc = mid+1**; otherwise goto step 9

**Step 9:** Check if key element is less than mid element is true, then initialize **high=mid-1** then goto step 6, if it false goto step10.

**Step 10:** Check if key element is greater than mid element is true, then initialize **low=mid+1** then goto step 6.

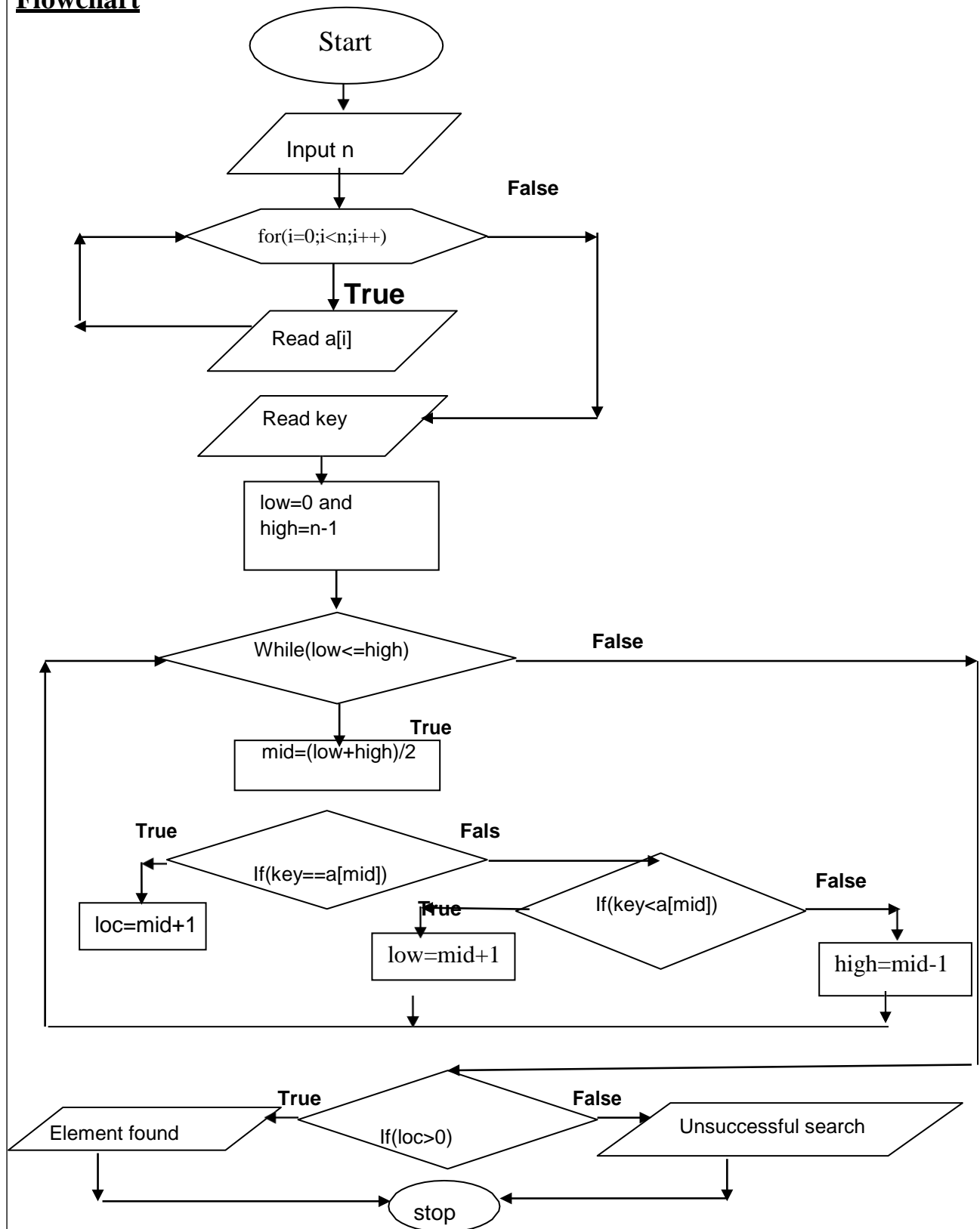
**Step 11:** Check if **loc** value is greater then **zero** then print the search is successful then goto step 13, otherwise goto step 12.

**Step 12:** print search is unsuccessful, then goto step 13.

**Step 13:** Stop

---

## Flowchart



---

## **Program**

```
#include<stdio.h>

void main()
{
    int n, a[100], i, key, high, low, mid, loc=-1;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    printf("Enter the elements of array in sorted order\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("Enter the key element to be searched\n");
    scanf("%d",&key);
    low=0;
    high=n-1;
    while(low<=high)
    {
        mid=(low+high)/2;
        if(key==a[mid])
        {
            loc = mid+1;
            break;
        }
        else
        {
            if(key<a[mid])
                high=mid-1;
            else
                low=mid+1;
        }
    }
    if(loc>0)
        printf("\n The element %d is found at %d ",key,loc);
    else
        printf("\nThe search is unsuccessful");
}
```

---



---

## **Output**

---

**8. Develop a program to sort the given set of N numbers using Bubble sort.**

**Algorithm**

**Input:** A unsorted array of n numbers

**Output:** sorted array

**Step 1:** START

**Step 2:** Read unsorted array n elements in to a[i], where i is the index value.

**Step 3:** Initialize index i=0.

**Step 4:** Check if i is less than n. if true, goto step 5. Otherwise goto step output array.

**Step 5:** initialize index j to zero.

**Step 6:** check if j is less than (n-i-1). If true goto step 7. Otherwise increment j and goto step 4.

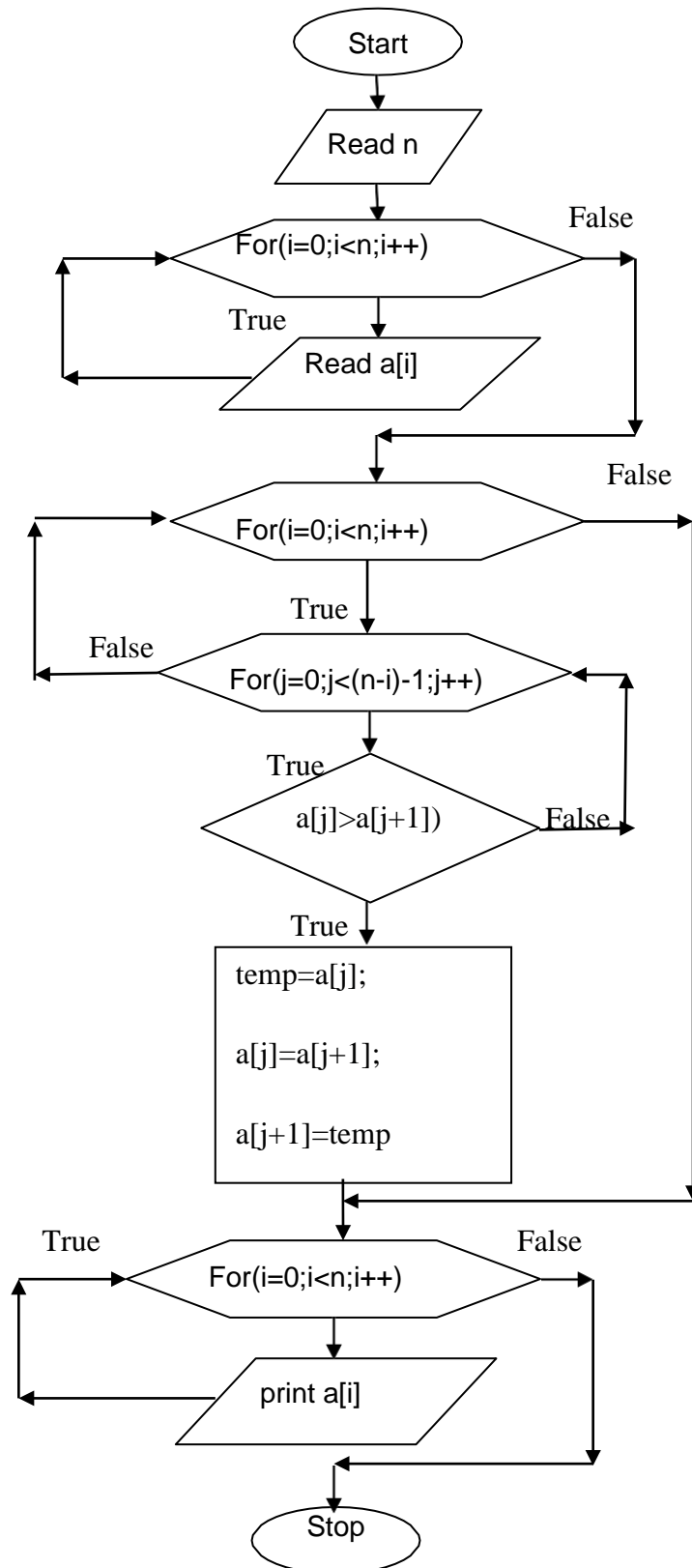
**Step 7:** inside the for loop check if a[j] is greater than a[j+1](i.e., adjacent elements are compared). If true swap elements using temporary variables. Otherwise goto step 6.

**Step 8:** Output the sorted array elements using for loop.

**Step 9:** Stop

---

## Flowchart



---

## **Program**

```
#include<stdio.h>

int main()
{
    int i,j,n,temp;
    int a[20];
    printf("enter the value of n");
    scanf("%d",&n);
    printf("Enter the numbers in unsorted order:\n");
    for(i=0;i<n;i++)
    scanf("%d", &a[i]);
    // bubble sort logic
    for(i=0;i<n;i++)
    {
        for(j=0;j<(n-i)-1;j++)
        {
            if( a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    printf("The sorted array is\n");
    for(i=0;i<n;i++)
    {
        printf("%d\n",a[i]);
    }
}
```

---

---

Output

---

|