C Programs to Begin

Program 1

Write a C program to display the message Hello World.

```
/* This program prints one-line message */
#include<stdio.h>
int main()
{
  printf("Hello World\n");
  return 0;
}
/* This program ... */ The symbols /* and */ delimit a comment.
```

Comments are ignored by the compiler, and are used to provide useful information for the reader of the program.

1. main( ) – C programs consist of one or more functions. One and *only* one of these functions must be called as main. The brackets following the word main indicate that it is a function and not a variable.
2. {} – braces surround the body of the function, which consists of one or more instructions (*statements*).
3. printf( ) – is a library function that is used to print on the standard output stream (usually the screen).
4. "HelloWorld\n" – is a string constant.
5. \n – is the newline character.
6. ; – a semicolon terminates a statement.
7. return0; – returns the value zero to the operating system.

C is case-sensitive, so the names of the functions (main and printf) must be typed in lower case as above.

Sample Output

```
Hello World
```

Program 2

Write a C program to show addition operation along with suitable comments.

```
#include<stdio.h>
void main()
{
  /*----------Declarations----------*/
  int number;
  float amount;
  /*----------Assignment Statement----------*/
  number=100;
  /*----------Computational Statements----------*/
  amount=30.75+75.35;
  printf("\n%d", number);
  printf("\n%5.2f\n", amount);
}
```

Sample Output

```
100
106.10
```

Program 3

Write a C program to convert temperature values from Fahrenheit to degree Celsius.

```
#include<stdio.h>
void main()
{
  float f, c;
  printf("\n Enter the temperature value in Fahrenheit: ");
  scanf("%f", &f);
  c=(f-32.0)/1.8;
  printf("\n The equivalent temperature value in degrees Celsius
  is: %.2f\n", c);
}
```

Sample Output

```
Enter the temperature value in Fahrenheit: 0
The equivalent temperature value in degrees Celsius is: -17.78
```

Program 4

Write a C program that reads an integer and checks whether it is odd or even.

```
#include<stdio.h>
void main()
{
  int num=0, remainder=0;
```

```
while(num!=-1)     /* while -1 not entered*/
{
  printf("Enter an integer (-1 to stop): ");
  scanf("%d", &num);
  if(num!=-1) /* ready to stop if -1 else...*/
  {
    remainder=num % 2;
    /* test for even/odd. If the modulus yields 0,it is even */
    if(remainder==0)
      printf("%d is an even number.\n", num);
    else
      printf("%d is an odd number.\n", num);
  }
 }/* -1 was entered */
 printf("%d is an odd number.\n", num);
 printf("You ask to stop! Thank you.\n");
}
```

Program 5

Write a C program to find the smallest of two numbers using function.

```
double minimum(double, double);       /* prototype of minimum() */
#include<stdio.h>
int main()
{
 printf("%f\n", minimum(1.23, 4.56));
 return 0;
}
double minimum(double x, double y)   /* definition of minimum() */
{
 if (x<y)
   return x;
 else
   return y;
}
```

Program 6

Write a C program that can read three integers from the user and then determine the smallest value among the three integers.

```
#include<stdio.h>
void main()
```

```
{
  int i, num[3], smallest=0;
  printf("Enter 3 integers separated by a space: ");
  for(i=0; i<3; i++)
  scanf("%d", &num[i]);
    smallest=num[0];    /* assign the 1st element to smallest */
  for(i=1; i<3; i++)
    if(num[i]<smallest)   /* compare the others and keep storing
                          the smallest */
      smallest=num[i];
      printf("The smallest number among ");
  for(i=0; i<=3; i++)     /* print the element */
  printf("%d%d ", num[i], smallest);
}
```

Program 7

Write a C program to find the area of a circle, given the radius.

```
#include<stdio.h>
#define PI 3.142
void main()
{
  float radius, area;
  printf("\n Enter the radius of a circle: ");
  scanf("%f", &radius);
    area=PI*radius*radius;
  printf("\n Area of a circle=%f\n", area);
}
```

Output

```
Enter the radius of a circle: 10
Area of a circle = 314.200012
```

```
Enter the radius of a circle: 0
Area of a circle = 0.000000
```

Program 8

Write a C program to read the marks, calculate the percentage and display the result.

```
#include<stdio.h>
void main()
```

```
{
 float mark1, mark2, percentage;
 printf("\n Enter the marks of first subject: \n\t");
 scanf("%f", &mark1);
 printf("\n Enter the marks of second subject: \n\t");
 scanf("%f", &mark2);
 percentage=((mark1+mark2)/200)*100;
 if(percentage>40)
   printf("\n Passed \n");
 else
   printf("\n Failed \n");
}
```

Verify Output

Program 9

Write a C program for investment problem.

```
#define PERIOD 3
#define PRINCIPLE 5000.00
#include<stdio.h>
void main()
{
  int year;
  float amount, value, inrate;
  amount=PRINCIPLE, inrate=0.11, year=0;
  while(year <PERIOD)
  {
    printf("%2d\t%8.2f\n", year, amount);
    value=amount+inrate*amount;
    year=year+1;
    amount=value;
  }
}
```

Verify Output

Program 10

Write a C program to print whether a given number is positive, negative or zero.

```
#include<stdio.h>
void main()
{
  int n;
  printf("\n Enter the number: ");
  scanf("%d",&n);
  if(n>0)
    printf("\n Number is positive");
  else if(n==0)
    printf("\n Number is zero");
  else
    printf("\n Number is negative");
}
```

Output

```
Enter the number: 10
Number is positive
--------------------------------
Enter the number: 0
Number is zero
--------------------------------
Enter the number: -10
Number is negative
```

Term-Work 1

Problem Definition

Design, develop and execute a program in C to find and output all the roots of a given quadratic equation for non-zero coefficients.

Objectives

- To map the paper–pen method of finding roots of a quadratic equation to a flowchart/ algorithm and write a C program for the same.
- To understand the usage of appropriate programming constructs.
- To test the program for all possible inputs.

Prerequisites for Coding

- *if...else* statement

```
if(test expression)
  {
    True block statement(s)
  }
else
  {
    False block statement(s)
  }
  Statement x;
```

If the test expression is true, then the true block statement(s) are executed; otherwise the false block statement(s) are executed. In any case, either true block or false block will be executed but not both. In both the cases, control is transferred subsequently to the statement x.

Sample Program

Program 1

This program will give an idea of using if statements in all different forms, that is, with else and without else.

```
#include<stdio.h>
void main()
{
  int cows=6;
  if(cows>1)
    printf("We have cows\n");
  if(cows>10)
    printf("loads of them!\n");
  else
    printf("Executing else part...!\n");
  if(cows==5)
  {
    printf("We have 5 cows\n");
  }
  else if( cows==6)
  {
    printf("We have 6 cows\n");
  }
}
```

Sample Output

```
We have cows
Executing else part...!
We have 6 cows
Press any key to continue . . .
```

Related Theory

A quadratic equation is an equation equivalent to one of the forms of $ax^2 + bx + c = 0$, where a, b and c are real numbers and $a \neq 0$. So if we have an equation in x and the highest power is 2, it is quadratic.

Example: $x^2 = 5x - 6$

$$x^2 = 5x - 6$$
$$-5x + 6 \quad -5x + 6$$
$$x^2 - 5x + 6 = 0$$
$$(x - 3)(x - 2) = 0$$

$$x - 3 = 0 \text{ or } x - 2 = 0$$

$x = 3$ or $x = 2$ are obtained by subtracting $5x$ and adding 6 to both sides to get 0 on right side. Use the Null Factor law and set each factor = 0 and solve.

Example: $2x^2 - 3x = 0$

$$2x^2 - 3x = 0$$
$$x(2x - 3) = 0$$
$$x = 0 \text{ or } 2x - 3 = 0$$
$$x = 0 \text{ or } x = \frac{3}{2}$$

Roots of quadratic equation $ax^2 + bx + c = 0$ can be computed using formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

First $r = \sqrt{b^2 - 4ac}$ is computed. Depending on the value of $d = b^2 - 4ac$ roots can be real or complex.

1. If $d > 0$, that is, if d value is positive, then roots are real and distinct. The two roots are computed as follows:

$$x1 = (-b + r)/(2.0 * a)$$
$$x2 = (-b - r)/(2.0 * a)$$

If $a = 6, b = 11$ and $c = -35$, then $d = (b^2 - 4ac)$. Here

$$d = (11*11) - (4*6*(-35))$$
$$= 961$$

Since $d > 0$, the roots are real and distinct.

2. If $d = 0$, then roots are real and equal.

$$x1 = x2 = -b/(2.0*a)$$

3. If $d < 0$, then roots are complex.

$$x1 = -b/(2.0*a)$$
$$x2 = r/(2.0*a)$$

Algorithm

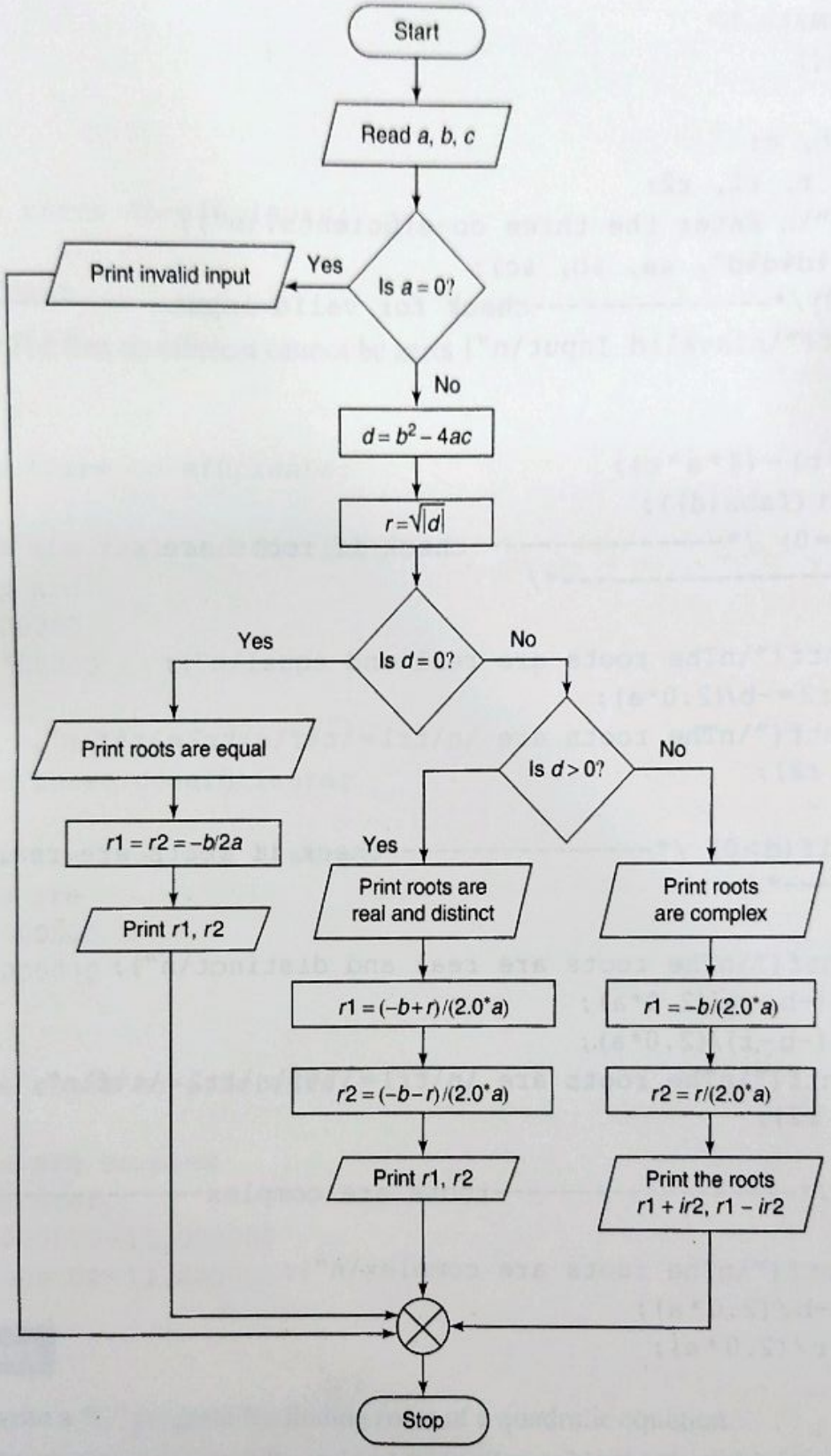
Start

Step 1: Input the co-efficients (a, b, c) of the quadratic equation.

Step 2: IF first co-efficient is zero
  Output "Invalid input"
ELSE
  $d \leftarrow b^2 - 4ac$
  $r \leftarrow \sqrt{|d|}$
IF the value of d is equal to zero
  Output "Roots are Equal"
  $r1 \leftarrow r2 \leftarrow -b/(2a)$
  Output $r1, r2$
ELSE IF the value of d is greater than zero
  Output "Roots are Real and Distinct"
  $r1 \leftarrow (-b + r)/(2a)$
  $r2 \leftarrow (-b - r)/(2a)$
  Output $r1, r2$
ELSE
  Output "Roots are Complex"
  $r1 \leftarrow -b/(2a)$
  $r2 \leftarrow r/(2a)$
  Print the roots
END IF
END IF

Stop

Flowchart


Start
Read a, b, c
Is a = 0?
Yes
Print invalid input
No
d = b² – 4ac
r = √|d|
Is d = 0?
Yes
No
Print roots are equal
r1 = r2 = –b/2a
Print r1, r2
Is d > 0?
No
Yes
Print roots are real and distinct
Print roots are complex
r1 = (–b+r)/(2.0*a)
r2 = (–b–r)/(2.0*a)
Print r1, r2
r1 = –b/(2.0*a)
r2 = r/(2.0*a)
Print the roots r1 + ir2, r1 – ir2
Stop

Program

```
#include<stdio.h>
#include<math.h>
void main()
{
 int a, b, c;
 float d, r, r1, r2;
 printf("\n Enter the three co-efficients:\n");
 scanf("%d%d%d", &a, &b, &c);
 if(a==0)/*--------------check for valid input---------------*/
  printf("\nInvalid Input\n");
 else
 {
  d=(b*b)-(4*a*c);
  r=sqrt(fabs(d));
  if(d==0) /*---------------check if roots are
  equal----------------*/
  {
   printf("\nThe roots are real and equal\n");
   r1=r2=-b/(2.0*a);
   printf("\nThe roots are \n\tr1=\t%f\n\tr2=\t%f\n",
   r1, r2);
  }
 else if(d>0) /*-----------------check if roots are real-------
 --------*/
 {
  printf("\nThe roots are real and distinct\n");
  r1=(-b+r)/(2.0*a);
  r2=(-b-r)/(2.0*a);
  printf("\nThe roots are \n\tr1=\t%f\n\tr2=\t%f\n",
  r1, r2);
 }
 else /*------------------roots are complex---------------*/
 {
  printf("\nThe roots are complex\n");
  r1=-b/(2.0*a);
  r2=r/(2.0*a);
```

```
        printf("\nThe roots are:\n\tr1=%f +i %f\n\tr2=%f-i
        %f\n",r1, r2, r1, r2);
    }
  }
}
```

Output

Sample 1:

```
Enter the three co-efficients:
0 1 2
Invalid Input
```

Conclusion: The first co-efficient cannot be zero.

Sample 2:

```
Enter the three co-efficients:
2 3 0
The roots are real and distinct
The roots are
  r1=0.000000
  r2=-1.500000
```

Sample 3:

```
Enter the three co-efficients:
1 5 6
The roots are real and distinct
The roots are
  r1=-2.000000
  r2=-3.000000
```

Sample 4:

```
Enter the three co-efficients:
1 2 2
The roots are complex
The roots are:
  r1=-1.000000+i1.000000
  r2=-1.000000-i1.000000
```

Outcomes

- Able to write a "C" program for finding roots of a quadratic equation.
- Capable to execute it successfully and test it for all possible input combinations.

Related Program

Program 1

Program in C to find and output all the roots of a given quadratic equation, for non-zero coefficients, ***using Switch statement.***

```
#include<stdio.h>
#include<math.h>
#include<process.h>
void main()
{
  int option;
  double a, b, c, d;
  double r,r1,r2;
  printf(" Quadratic Equation ax^2+bx+c=0 \n");
  printf("*************************************\n");

  /* Get a, b, and c from user */
  printf("\na=");
  scanf("%lf", &a);
  printf("b=");
  scanf("%lf", &b);
  printf("c=");
  scanf("%lf", &c);

  if(a==0) /* check for valid input */
  {
    printf("\n Invalid Input \n");
    exit(0);
  }

  /* Calculate determinant */
  d=(b*b)-(4*a*c);
  r=sqrt(fabs(d));
  if(d==0)
    option=1;
  else if(d>0)
    option=2;
  else
    option=3;
  /* Use switch statement */
```

```
  switch(option)
  {
    case 1: printf("\nThe roots are real and equal\n");
      r1=r2=-b/(2.0*a);
      printf("\nThe roots are \n\tr1=\t%f\n\tr2 =\t%f\n", r1, r2);
      break;
    case 2: printf("\nThe roots are real and distinct\n");
      r1=(-b+r)/(2.0*a);
      r2=(-b-r)/(2.0*a);
      printf("\nThe roots are \n\tr1=\t%f\n\tr2=\t%f\n",
      r1, r2);
      break;
    case 3: printf("\nThe roots are complex\n");
      r1=-b/(2.0*a);
      r2=r/(2.0*a);
      printf("\nThe roots are:\n\tr1=%f +i %f\n\tr2=%f -i
      %f\n",r1, r2, r1, r2);
      break;
  }/* End Switch */
}/* End program */
```

Output

```
1.
Quadratic Equation ax^2+bx+c=0
****************************************
a=0
b=1
c=2
Invalid Input

2.
Quadratic Equation ax^2+bx+c=0
****************************************
a=1
b=2
c=1
The roots are real and equal
The roots are
  r1=-1.000000
  r2=-1.000000
```

```
3.
Quadratic Equation ax^2+bx+c=0
****************************************
a=1
b=3
c=2
The roots are real and distinct
The roots are
  r1=-1.000000
  r2=-2.000000

4.
Quadratic Equation ax^2+bx+c=0
****************************************
a=1
b=2
c=1
The roots are complex
The roots are:
  r1=-1.000000+i1.000000
  r2=-1.000000-i1.000000
```

Term-Work 2

Problem Definition

Design, develop and execute a program in C to implement Euclid's algorithm to find the GCD and LCM of two integers and to output the results along with the given integers.

Objectives

- To learn Euclid's algorithm.
- To write a C program to execute Euclid's algorithm for any given set of two integers.
- To learn to correlate the paper–pencil method with the developed algorithm and test it for all possible types of input.

Prerequisites for Coding

Looping Constructs

There are circumstances where you want to do the same thing many times. For instance, you want to print the same word 10 times. You could type 10 printf functions, but it is easier to use a loop. The only thing you have to do is to set up a loop that executes the same printf function 10 times. There are three basic types of loops which are as follows:

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

```
while (condition)
    {
      Action1 /* until condition is true action1 is executed*/
    }
    Action2 /* if condition fails action2 is executed*/
```

In programming languages, a while loop is a *control flow statement* that allows code to be executed repeatedly based on a given *Boolean* condition. A while loop can be thought of as a repeating *if statement*. A *while* construct consists of a block of code and a condition. The condition is evaluated, and if the condition is *true*, the code within the block is executed. This

repeats until the condition becomes *false*. Because *while* loop checks the condition before the block is executed, the control structure is known as a pre-test loop (entry-controlled loop).

Sample Program: While Loop

Program 1

Program to print the message "Hello" 10 times. This program will give an idea of how while loop works.

```
#include <stdio.h>
void main()
{
  int i=10;
  while (i > 0)
  {
    printf("Hello\n");
          i=i - 1;
  }
}
```

Sample Output

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

Related Theory

The **Euclidean algorithm** (also called **Euclid's algorithm**) is an efficient method for computing the greatest common divisor (GCD), also known as the greatest common factor (GCF) or highest common factor (HCF). It is named after the Greek mathematician Euclid.

The GCD of two numbers is the largest number that divides both of them without leaving a remainder.

Example

Calculation of the GCD of 18 and 48:
The divisors/factors of 18 are 1, 2, 3, 6, 9 and 18.
The divisors/factors of 48 are 1, 2, 3, 4, 6, 8, 12, 16, 24 and 48.
So, the common divisors/factors of 18 and 48 are 1, 2, 3 and 6.
We then have GCD(18, 48) = 6.

However, this method is not generally applied to large numbers as it is time consuming; hence we use Euclid's algorithm.

Example

Calculation of GCD of 142 and 38

$$142 = 38 \times 3 + 28$$
$$38 = 28 \times 1 + 10$$
$$28 = 10 \times 2 + 8$$
$$10 = 8 \times 1 + 2$$
$$8 = 2 \times 4 + 0$$
$$\text{GCD}(142, 38) = 2$$

The algorithm is based on the following two observations:

- If b divides a, then $\text{GCD}(a, b) = b$.
- This is indeed so because no number (b, in particular) may have a divisor greater than the number itself.
- If $a = bt + r$, for integers t and r, then $\text{GCD}(a, b) = \text{GCD}(b, r)$.

Euclidean Algorithm

1. Divide the first number by the second one.
2. If remainder is zero, second number is the GCD.
3. Otherwise, repeat steps 1 and 2 where second number becomes first and remainder becomes the second number.

Let's look at an example to find the GCD of 40 and 64.

- First number = 40, second number = 64, remainder = 40
- First number = 64, second number = 40, remainder = 24

- First number = 40, second number = 24, remainder = 16
- First number = 24, second number = 16, remainder = 8
- First number = 16, second number = 8, remainder = 0

Hence GCD = 8.

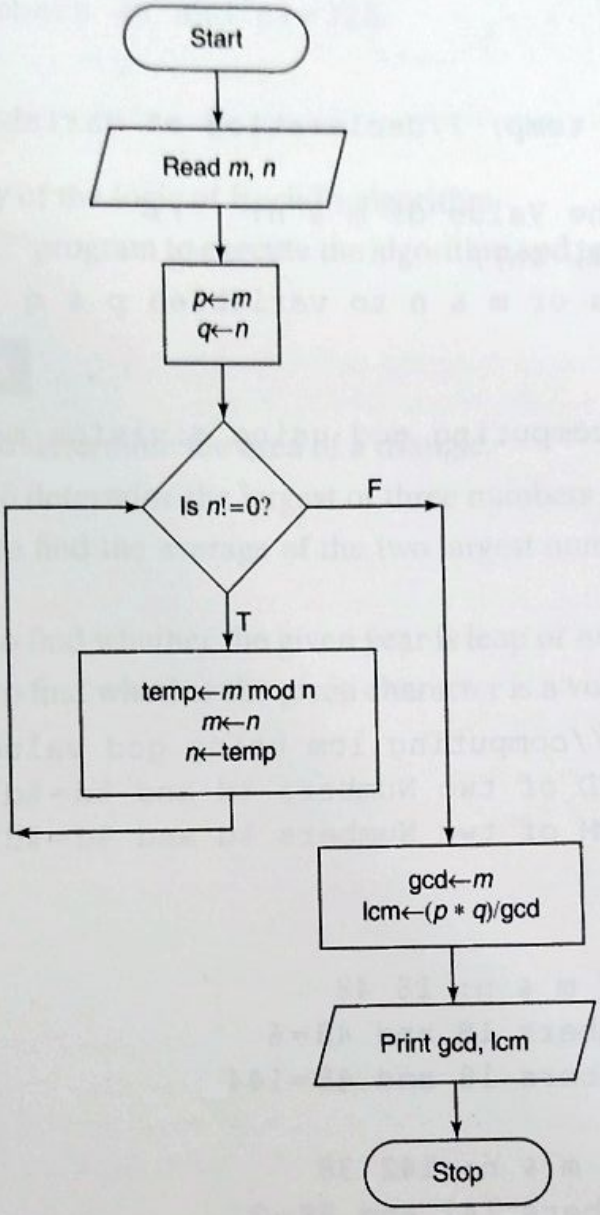
LCM is obtained using GCD as **LCM = product of given two numbers/GCD.**
Using same example given above, we can find the LCM (40, 64) as follows:

$$LCM = (40 * 64)/8 = 320$$

Algorithm

Start
Step 1: Read m, n [two numbers], $p \leftarrow m, q \leftarrow n$
Step 2: WHILE $n! = 0$ DO
temp $\leftarrow m$ mod n
$m \leftarrow n$
$n \leftarrow$ temp
END WHILE
Step 3: gcd $\leftarrow m$
lcm $\leftarrow (p*q)$/gcd
Step 4: Print gcd, lcm
Stop

Flowchart


Start
Read m, n
p←m
q←n
Is n!=0?
F
T
temp←m mod n
m←n
n←temp
gcd←m
lcm←(p * q)/gcd
Print gcd, lcm
Stop

Program

```
#include<stdio.h>
void main()
{
  int m, n, p, q, temp; //declaration of variables
  int gcd, lcm;
  printf("Enter the Value of m & n:  ");
  scanf("%d%d", &m, &n);
  //copying values of m & n to variables p & q
  p=m;
  q=n;
  while(n!=0) // computing gcd using division method
  {
    temp=m%n;
    m=n;
    n=temp;
  }
  gcd=m;
  lcm=(p*q)/gcd; //computing lcm using gcd value
  printf("\nThe GCD of two Numbers %d and %d=%d\n",p,q,gcd);
  printf("\nThe LCM of two Numbers %d and %d=%d\n",p,q,lcm);
}
```

Output

```
Enter the Value of m & n: 18 48
The GCD of two Numbers 18 and 48=6
The LCM of two Numbers 18 and 48=144

Enter the Value of m & n: 142 38
The GCD of two Numbers 142 and 38=2
The LCM of two Numbers 142 and 38=2698

Enter the Value of m & n: 5 0
The GCD of two Numbers 5 and 0=5
The LCM of two Numbers 5 and 0=0
```

Note: If any one of the two numbers is 0, then the other number is itself the GCD. Since (5*0) is 0, LCM is 0.

```
Enter the Value of m & n: 40 64
The GCD of two Numbers 40 and 64=8
The LCM of two Numbers 40 and 64=320
```

Outcomes

- To verify the efficiency of the logic of Euclid's algorithm.
- To be able to write a "C" program to execute the algorithm and test it for all possible inputs.

Practice Programs

1. Write a C program to determine the area of a triangle.
2. Write a C program to determine the largest of three numbers a, b, c.
3. Write a C program to find the average of the two largest numbers of the three entered numbers a, b, c.
4. Write a C program to find whether the given year is leap or not.
5. Write a C Program to find whether the given character is a vowel or a consonant.

Term-Work 3

Problem Definition

Design, develop and execute a program in C to reverse a given four-digit integer number and check whether it is a palindrome or not. Output the given number with a suitable message.

Objectives

- To teach the logic to reverse a given four-digit number.
- To understand the usage of "/" (division), "%"(modulus) and "||"(Logical OR) operators in writing the program efficiently.
- To identify the application of palindromes.

Prerequisites for Coding

Division Operator "/"

When applied to integers, the division operator "/" accepts integer part and discards the fractional part of the quotient, so 1/2 is 0 and 7/4 is 1. But when either operand is a floating-point quantity (type `float` or `double`), the division operator yields a floating-point result, with a potentially non-zero fractional part. So 1/2.0 is 0.5 and 7.0/4.0 is 1.75.

Modulus Operator "%"

The modulus operator "%" gives you the remainder when two integers are divided: 1% 2 is 1; 7 % 4 is 3. (The modulus operator can only be applied to integers.)

Logical Operator OR "||"

Logical operators are used to test more than one condition and make decisions.

For example, $a < 10 \| a > 100$. An expression of this kind, which combines two or more relational expressions, is termed as a logical expression. These expressions yield a value of true or false, as shown below:

*Op*1	*Op*2	*Op*1 || *Op*2
True	True	True
True	False	True
False	True	True
False	False	False

Related Theory: Logic Used

A ***palindrome*** is a word or a sentence that reads the same forward as it does backward. The word "palindrome" was coined from Greek roots *palin* (*πάλιν*; "again") and *dromos* (*δρόμος*; "way, direction") by English writer Ben Jonson in the 17th century. Neither spaces nor punctuation are usually taken into consideration when constructing sentences that are palindromes – one of the most famous palindromes is "A man, a plan, a canal, Panama" – but when the spaces are properly positioned as well, so much the better. An example would be the also famous palindrome "Able was I ere I saw Elba" purportedly spoken by Napoleon, referring to his first sighting of Elba, the island where the British exiled him.

Palindromes are a type of ***palingram*** called ***letter palingrams***. A ***palingram*** is a sentence in which the letters, syllables or words read the same backward as they do forward. The sentence, "He was, was he?" is a word palingram, because the words can be placed in reverse order and still read the same. The sentence, "I did, did I?" is not only a word palingram but a letter palingram (or palindrome) as well. In the same way, integers may be palindromes. For example, 1221, 3333, ...

Sample Program

Program 1

Program to find the largest of three numbers. This program will give an idea of logical operators.

```
#include<stdio.h>
void main()
{
  int a, b, c;
  int big;
  printf("Enter any three numbers: ");
  scanf("%d%d%d", &a, &b, &c);
  if(a>b && a>c)
    big=a;
```

```
  else if(b>c)
    big=b;
  else
    big=c;
  printf("Largest number is: %d", big);
}
```

Sample Outputs

```
1. Enter any three numbers: 345 56 2
   Largest number is: 345
2. Enter any three numbers: 13 25 6
   Largest number is: 25
```

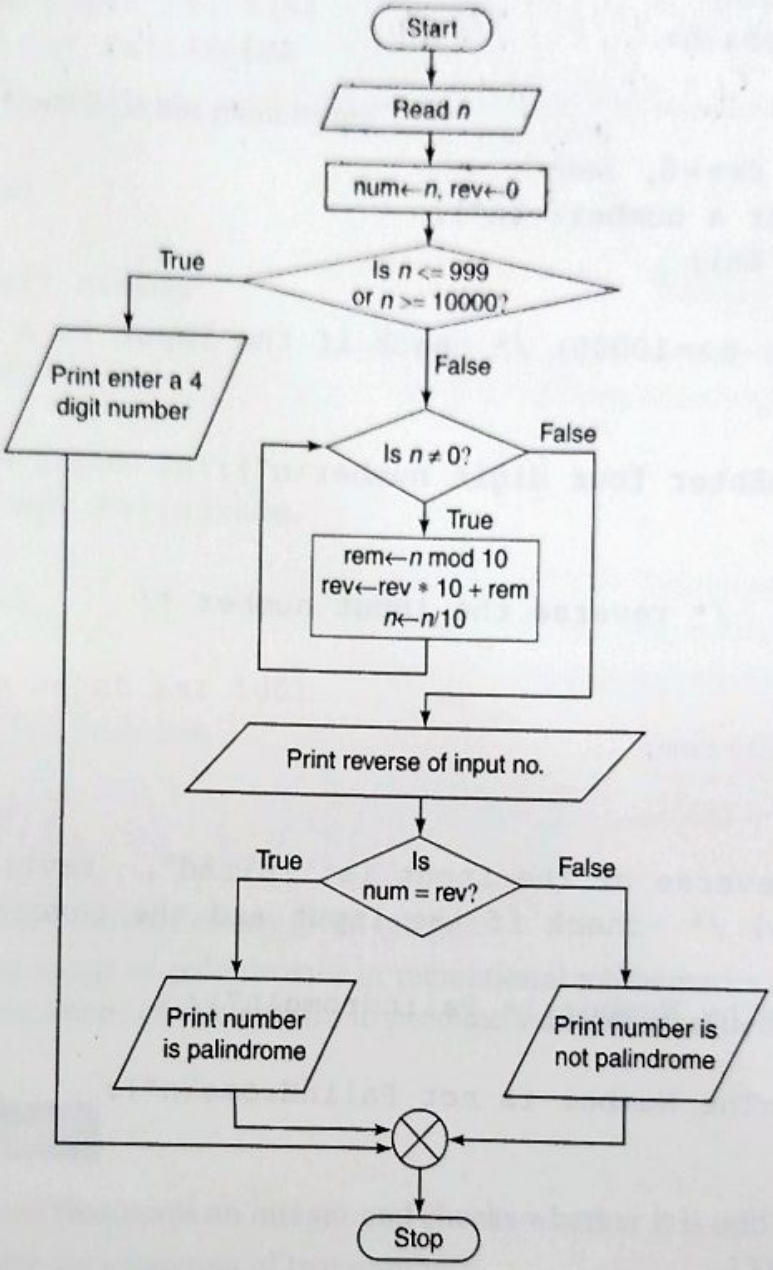
Algorithm

Start
Step 1: Read a number n
Step 2: If n is a four-digit number go to step 3
Else error in the input.
Step 3: Reverse the number
Step 4: If reversed number equals original number
Print number is palindrome
Else Print number is not palindrome
Stop

Algorithm to Reverse a Number

Start
Step 1: rev $\leftarrow$ 0
Step 2: rem $\leftarrow$ n mod 10
Step 3: rev $\leftarrow$ rev * 10+rem
Step 4: $n \leftarrow n/10$
Step 5: Repeat steps 2–4 until n becomes zero.
Stop

Flowchart


Start
Read n
num←n, rev←0
Is n <= 999 or n >= 10000?
True
False
Print enter a 4 digit number
Is n ≠ 0?
False
True
rem←n mod 10
rev←rev * 10 + rem
n←n/10
Print reverse of input no.
Is num = rev?
True
False
Print number is palindrome
Print number is not palindrome
Stop

Program

```
#include<stdio.h>
#include<process.h>
void main()
{
  int n, num, rev=0, rem;
  printf("Enter a number: \n");
  scanf("%d", &n);
  num = n;
  if(n<=999 || n>=10000) /* check if the input is a 4 digit
  number */
  {
    printf("\nEnter four digit number\n");
    exit(0);
  }
  while(n!=0)  /* reverse the input number */
  {
    rem=n%10;
    rev=rev*10+rem;
    n=n/10;
  }
  printf("\nReverse of the input is: \n\t%d",  rev);
  if(num==rev) /*  check if the input and the reversed numbers
  are same */
    printf("\nThe Number is Palindrome\n");
  else
    printf("\nThe Number is not Palindrome\n");
}
```

Output

```
Enter a number:
121
Enter four digit number

Enter a number:
12221
Enter four digit number
```

Note: From the above two sample outputs, it can be concluded that the program will accept only four-digit numbers.

```
Enter a number:
4578
Reverse of the input is: 8754
The Number is not Palindrome
```

Conclusion: "The number is not palindrome".

```
Enter a number:
0001
Enter four digit number

Enter a number:
1000
Reverse of the input is: 1
The Number is not Palindrome

Enter a number:
01001
Reverse of the input is: 1001
The Number is Palindrome
```

Outcomes

- To be able to find palindrome of any given integer.
- To understand the usage of palindromes in recreational mathematics, genomes and recognize a specific sequence of nucleotides to produce a double-stranded cut in the DNA.

Practice Programs

1. Write a C program that reads an integer and checks whether it is odd or even.
2. Write a C program for swapping of two numbers.
3. Write a C program to find the largest of *n* numbers in C.
4. Write a C program to split a number into digits in C programming.
5. Write a C program to count the number of digits in a number.

Term-Work 4

Problem Definition

Design, develop and execute a program in C to evaluate the given polynomial $f(x)=a_4x^4+a_3x^3+a_2x^2+a_1x+a_0$ for a given value of x and the coefficients using Horner's method.

Objectives

- To understand the underlying logic of Horner's method.
- To convert the mathematical logic to an algorithm and write a C code.
- To test the working of the program with valid and invalid inputs.

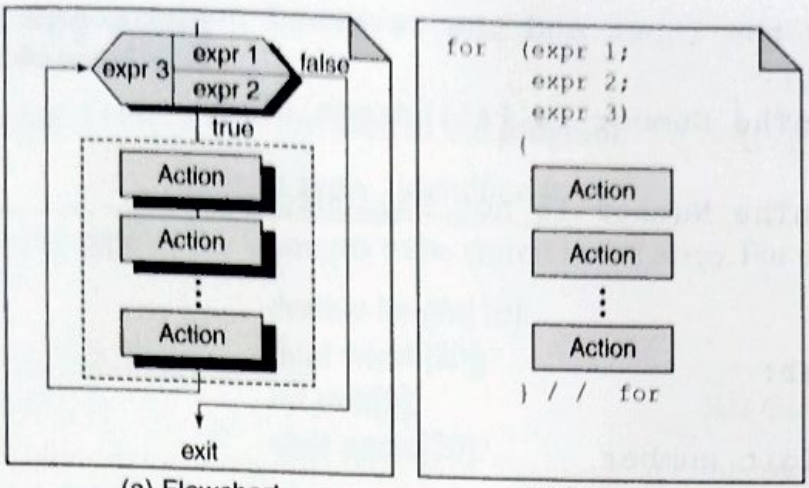
Prerequisites for Coding

The for loop is another entry-controlled loop that provides a more concise loop control structure. The syntax of for loop is as below:

```
for(initialization; test-condition; expression)
{
   Body of the loop;
}
```

Execution of "for loop" is as follows:

1. Initialization of loop control variables is done first, using assignment statements.
2. Value of the control variable is tested using the test condition, which is a relational expression that determines when the loop will exit. If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the expression is evaluated and the condition is tested. This process continues till the value of the control variable fails to satisfy the test condition.



(a) Flowchart (b) C language

Sample Program

Program in C to reverse a given four-digit integer number and check whether it is a palindrome or not ***using for loop***. Output the given number with a suitable message.

```
#include<stdio.h>
#include<process.h>
void main()
{
  int n, num, rev=0, rem, i;
  printf("Enter a number: \n");
  scanf("%d", &n);
  num=n;

  /* check if the input is a 4 digit number */
  if(n<=999||n>=10000)
  {
    printf("\nEnter four digit number\n");
    exit(0);
  }

  for(i=n; i!=0;)
  {
    rem=i%10;
    rev=rev*10+rem;
    i=i/10;
  }

  printf("\nReverse of the input is: \n\t%d", rev);
```

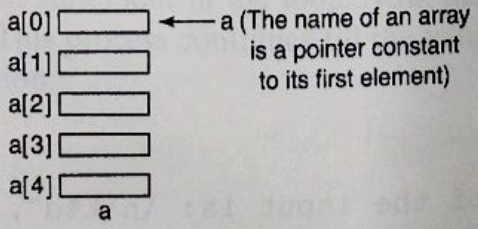
```
  /* check if the input and the reversed numbers are same */
  if(num==rev)
    printf("\nThe Number is Palindrome\n");
  else
    printf("\nThe Number is not Palindrome\n");
}
```

Sample Output

```
Enter a number:
121
Enter four digit number

Enter a number:
12221
Enter four digit number

Enter a number:
4578
Reverse of the input is: 8754
The Number is not Palindrome

Enter a number:
1221
Reverse of the input is: 1221
The Number is Palindrome

Enter a number:
1000
Reverse of the input is: 1
The Number is not Palindrome
```

Array

An array is a collection of same type of elements under the same identifier referenced by an index number. Arrays are efficient and useful for performing operations such as sorting, searching, etc.



Declaration of an Array

Arrays must be declared before they are used in the program.

Syntax: type identifier[size];

Here type specifies the type of the elements to be stored in the array. For example,

```
double height[10];
float width[20];
int min[9];
char name[20];
```

In C language, arrays start at position 0. The elements of the array occupy adjacent locations in memory. Any item in the array can be accessed through its index. For example,

```
m = height[0];
```

The variable *m* will be assigned the value of the first item of the array height.

The program segment below declares an array of five integers and prints all the elements of the array.

```
int myArray[5] = {1, 2, 3, 4, 5};

/* To print all the elements of the array
for (int i=0; i<5; i++)
{
  printf("%d", myArray[i]);
}
```

Related Theory: Logic Used

General form of a polynomial is

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a_1 x + a_0$$

Consider an example

$$3x^3 + 4x^2 + 7x + 8$$

Generally a polynomial is evaluated as

$$3 * x * x * x + 4 * x * x + 7 * x + 8$$

Hence

Number of multiplications = 6

Number of additions = 3

Horner's Methodology

- Polynomial expression

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a_1 x + a_0$$

Same can be written in another form as

$$P(x) = (a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x \cdot a_n))))$$

- In a parenthesized expression, always inner parenthesis is evaluated first, hence $(a_{n-1} + x \cdot a_n)$ is evaluated first.
- So we can take a constant a_n as initial value of "*result*" variable.

$result = a_n$

then, $P(x) = (a_0 + x(a_1 + x(a_2 + \cdots + X(a_{n-1} + x \cdot result))))$

Inner term as $\quad result = (a_{n-1} + x \cdot result)$

Next term as $\quad result = (a_{n-2} + x \cdot result)$

Next term as $\quad result = (a_{n-3} + x \cdot result)$

.

.

.

Outer term as $\quad result = (a_0 + x \cdot result)$

Generally result $= a_i + x \cdot$ result where i is from $n - 1$ to 0.

Example

Consider as an example

$$3x^3 + 4x^2 + 7x + 8$$

It can be written in another form as

$$(8 + x * (7 + x * (4 + x * 3)))$$

Hence

Number of multiplications = 3

Number of additions = 3.

Number of multiplications in the conventional method are more than the number of multiplications in the Horner's method. Hence, Horner's method is efficient.

Important steps in evaluation of polynomial are as follows:

- Read in all the coefficients into array, say a. Initialize a variable like *result* to a_n.
- Then repeatedly execute the statement

$$result = result * x + a[i],$$

for all values of i from $n - 1$ to 0 in steps of -1.

Sample Programs

Program 1

Program to print the table for 8 in one column. This program will give an idea of using the for loop and also to have correct indentations using escape sequences.

```
#include<stdio.h>
void main()
{
  int i, j=8;
  printf("Times 8 Table\n");
  for(i=0; i<=12; i=i+1)
  {
    printf("%d x %d=%d\n", j, i, j*i);
  }
  printf("\n");
}
```

Sample Output

```
Times 8 Table
8×0=0
8×1=8
8×2=16
8×3=24
8×4=32
8×5=40
8×6=48
8×7=56
8×8=64
8×9=72
8×10=80
8×11=88
8×12=96
```

Program 2

Program to find the sum and average of array elements. This program will give an idea of arrays, reading and printing elements of the arrays and performing addition and division operation.

```
#include<stdio.h>
void main()
{
  int a[5], i, sum=0;
  printf("Enter five elements for an array \n");
  for(i=0; i<5; i++)
    scanf("%d", &a[i]);
  printf("The list elements are \n");
  for(i=0; i<5; i++)
    printf("%d\t", a[i]);
  for(i=0; i<5; i++)
    sum=sum+a[i];
  printf("\nThe sum of the elements of the array is: %d\n", sum);
  sum=sum/5;
  printf("\nThe average of the elements of the array is: %d\n",
  sum);
}
```

Sample Output

```
Enter five elements for an array
1 2 3 4 5
The list elements are
1   2   3   4   5
The sum of the elements of the array is: 15
The average of the elements of the array is: 3
```

Program 3

Program to copy elements of an array into another array. This program will give an idea of arrays. There is no such statement in C language that can directly copy an array into another array. So we have to copy each item separately into another array.

```
#include<stdio.h>
int main()
{
  int iMarks[4], i, j;
  short newMarks[4];
  iMarks[0]=78;
```

```
  iMarks[1]=64;
  iMarks[2]=66;
  iMarks[3]=74;
  for(i=0; i<4; i++)
    newMarks[i]=iMarks[i];
  for(j=0; j<4; j++)
    printf("%d\n",  newMarks[j]);
}
```

Sample Output

```
78
64
66
74
```

Algorithm

Start

Step 1: Input *a* number *n*

Step 2: FOR $i \leftarrow 0$ to n in steps of 1
Input the elements of array $a[i]$
END FOR

Step 3: Input the value of x

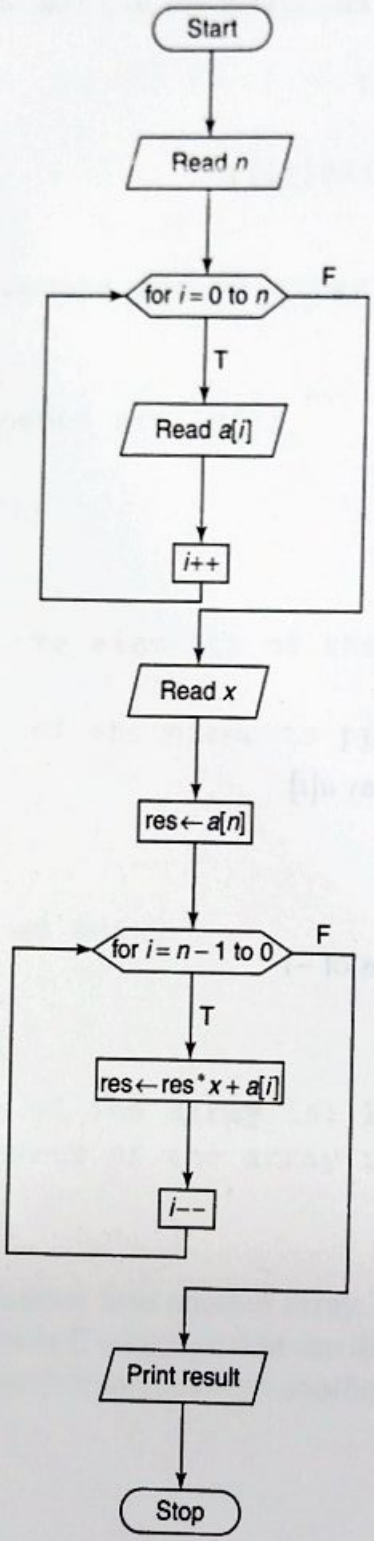
Step 4: res $\leftarrow a[n]$

Step 5: FOR $i \leftarrow n-1$ to 0 in steps of -1
res $\leftarrow$ res * $x + a[i]$
END FOR

Step 6: Output the value of res

Stop

Flowchart


Start
Read n
for i = 0 to n
F
T
Read a[i]
i++
Read x
res ← a[n]
for i = n – 1 to 0
F
T
res ← res * x + a[i]
i--
Print result
Stop

Program

```
#include<stdio.h>
void main()
{
 int a[40];
 int x, n, i, res;
 printf("\nEnter maximum power of x:\n\t");
 scanf("%d", &n); /* Read the maximum size of array */
 printf("\nEnter the coefficients of the polynomial:\n");
 /*-------------read elements of the array-------------*/
 for(i=0; i<=n; i++)
 {
  printf("\nEnter coefficient of X^%d=\t", i);
  scanf("%d", &a[i]);
 }
 printf("\nEnter the value of X=\t");
 scanf("%d", &x);
 /*--------------Compute the value of the
 polynomial---------------*/
 res=a[n];
 for(i=n-1; i>=0; i--)
   res=res*x+a[i];
 printf("Value of polynomial=\t%d\n", res); /* print the result
 of the polynomial */
}
```

Output

```
Enter maximum power of x: 2
Enter the coefficients of the polynomial:
Enter coefficient of X^0=1
Enter coefficient of X^1=2
Enter coefficient of X^2=3
Enter the value of X=2
Value of polynomial=17

Enter maximum power of x: 5
Enter the coefficients of the polynomial:
Enter coefficient of X^0=4
Enter coefficient of X^1=2
Enter coefficient of X^2=6
```

```
Enter coefficient of X^3=1
Enter coefficient of X^4=7
Enter coefficient of X^5=8
Enter the value of X=4
Value of polynomial=10156

Enter maximum power of x: 1
Enter the coefficients of the polynomial:
Enter coefficient of X^0=2
Enter coefficient of X^1=3
Enter the value of X=1
Value of polynomial=5
```

Outcomes

- To verify the efficiency of the evaluation of given polynomials.
- Should be able to use appropriate looping constructs.
- Capable of executing the program with valid and invalid inputs.

Practice Programs

1. Write a C program to print out all integers from 5 down to –5.
2. Write a C program to print out the squares of the first 10 integers. Ask the user for two integers, first a small one then a larger one. Multiply these two integers by doing repeated addition. For example, if we were to obtain 3 and 5 from the user, then we would find their product by adding the larger one (5) three times.
3. Write a simple C program that uses arrays to print out number of employees having salary more than 3000.
4. Write a C program that will demonstrate how to read, write and traverse the integer arrays.
5. Write a C program to ask the user for an integer and print out that integer's multiplication table.

Term-Work 5

Problem Definition

Design, develop and execute a program in C to copy its input to its output, replacing each string of one or more blanks by a single blank.

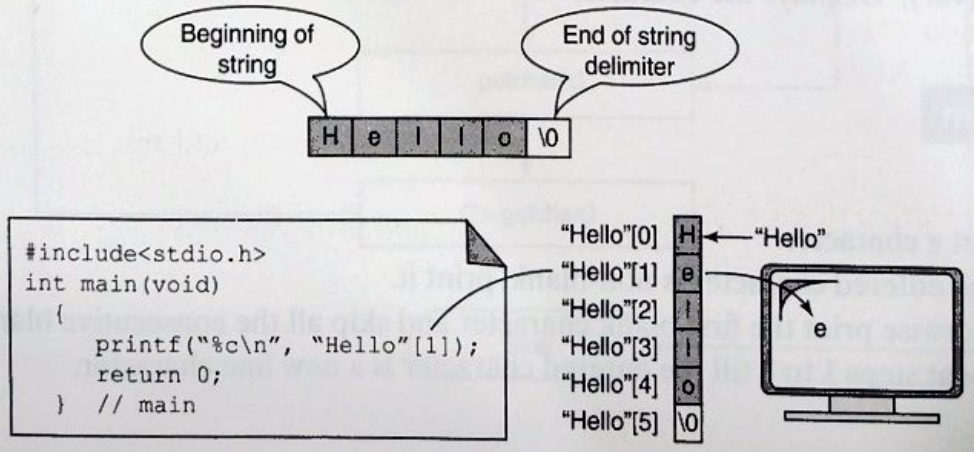
Objectives

- To handle input, output operations using the built-in function getchar() and putchar().
- To format the text.

Related Theory

What is string?

String in C is represented by array of characters. The end of the string is marked with a special character, the null character, which is simply the character with the value 0. In the ASCII character set, the null character is named NULL. The null or string-terminating character is represented by another character escape sequence'\0'.



Prerequisites for Coding

getchar function: getchar();

getchar function reads a single character from standard input. It takes no parameters and it returns the input character. In general, a reference to the getchar function is written as

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

For example,

```
char c;
c = getchar();
```

The second line causes a single character to be entered from the standard input device and then assigned to *c*. If an end-of-file condition is encountered when reading a character with the getchar function, the value of the symbolic constant EOF will automatically be returned.

This function can also be used to read multi-character strings, by reading one character at a time within a multi-pass loop.

putchar function: putchar(variable | constant);

The standard C function that prints or displays a single character by sending it to standard output is called putchar. This function takes one argument, which is the character to be displayed.

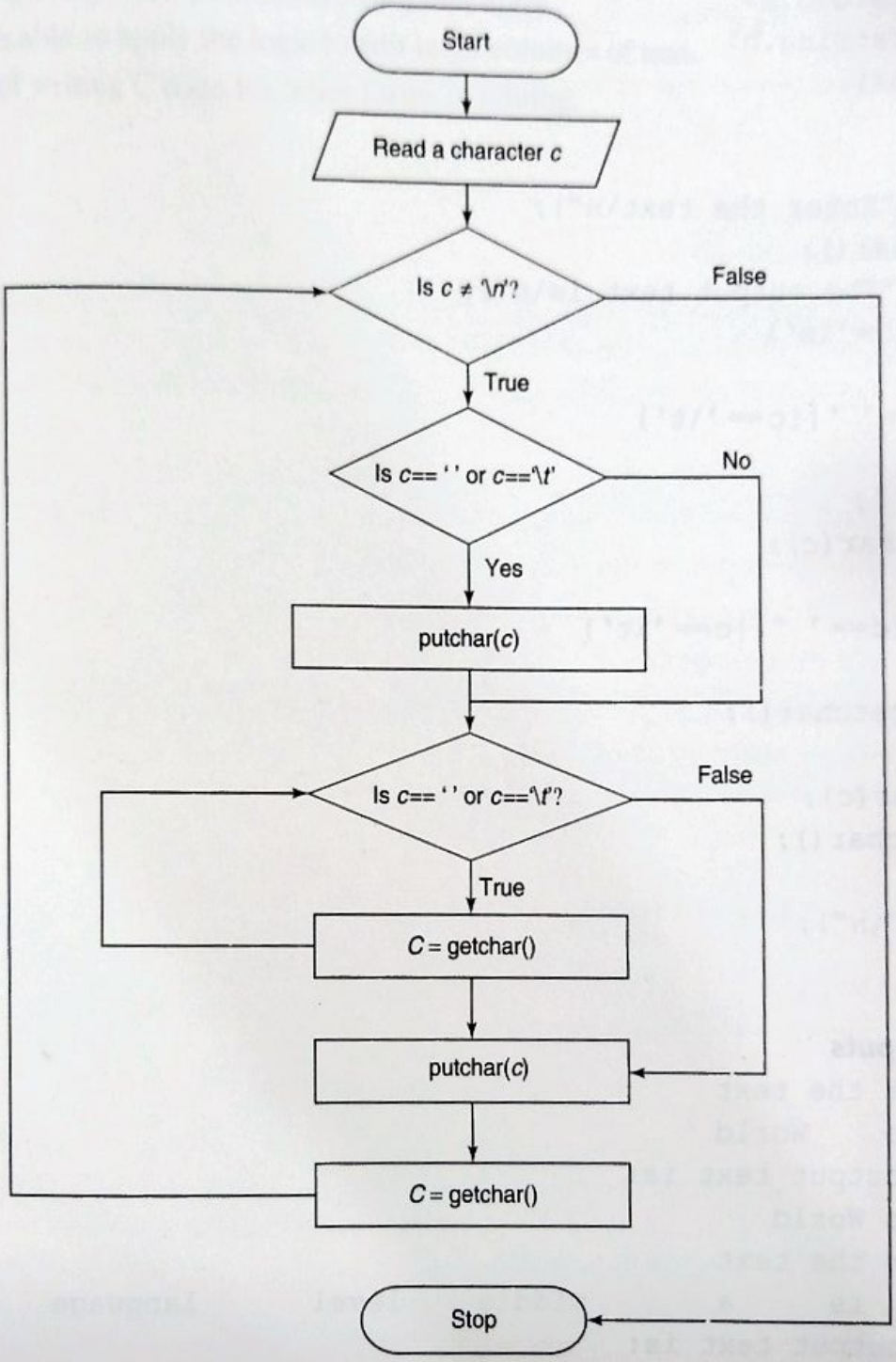
For example,

1. putchar ('N'); displays the character N.
2. char var = '*';
   putchar(var); Displays the character *.

Algorithm

Start
Step 1: Input a character.
Step 2: If the entered character is non-blank, print it.
Step 3: Otherwise print the first blank character and skip all the consecutive blanks.
Step 4: Repeat steps 1 to 3 till the entered character is a new line character.
Stop

Flowchart


Start
Read a character c
Is c ≠ '\n'?
False
True
Is c== ' ' or c=='\t'
No
Yes
putchar(c)
Is c== ' ' or c=='\t'?
False
True
C = getchar()
putchar(c)
C = getchar()
Stop

Program

```
#include<stdio.h>
#include<string.h>
void main()
{
  char c;
  printf("Enter the text\n");
  c=getchar();
  printf("The output text is\n");
  while(c!='\n')
  {
    if(c==' '||c=='\t')
    {
      c=' ';
      putchar(c);
    }
    while(c==' '||c=='\t')
    {
       c=getchar();
    }
    putchar(c);
    c=getchar();
  }
  printf("\n");
}
```

Sample Outputs

1. Enter the text
   Hello World
   The output text is:
   Hello World
2. Enter the text
   C is a middle level language
   The output text is:
   C is a middle level language

Outcomes

- Capable of using built-in standard string functions.
- Should be able to apply the logic to edit large volumes of text.
- Capable of writing C code for other forms of editing.

Term-Work 6

Problem Definition

Design, develop and execute a C program to input N integer numbers in ascending order into a single-dimensional array, and then to perform a binary search for a given key integer number and report success or failure in the form of a suitable message.

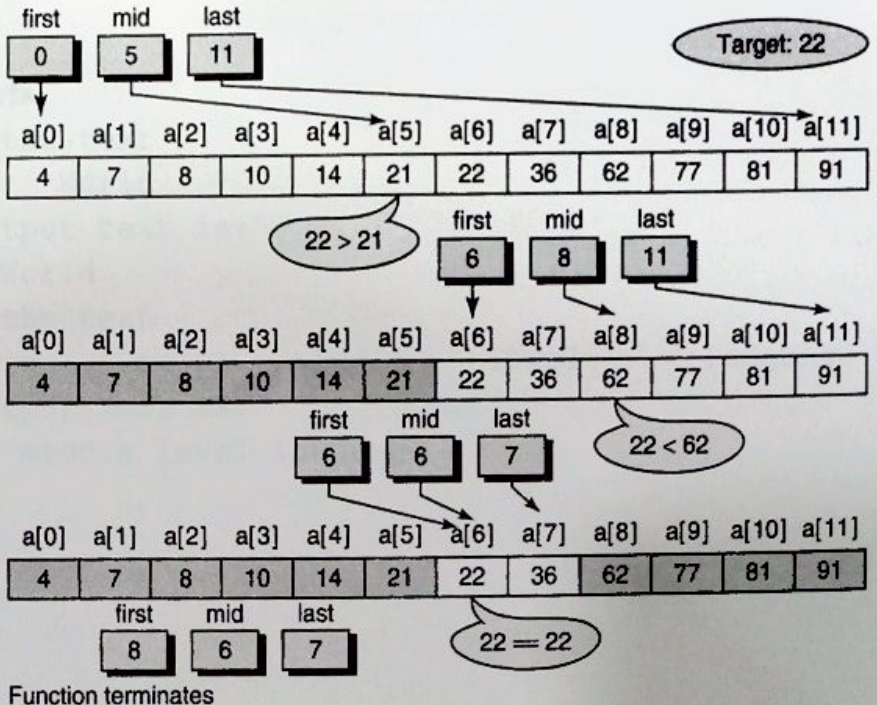
Objectives

- To understand the syntax, working and use of arrays.
- To learn the difference between linear and binary searching techniques.
- To implement binary search.

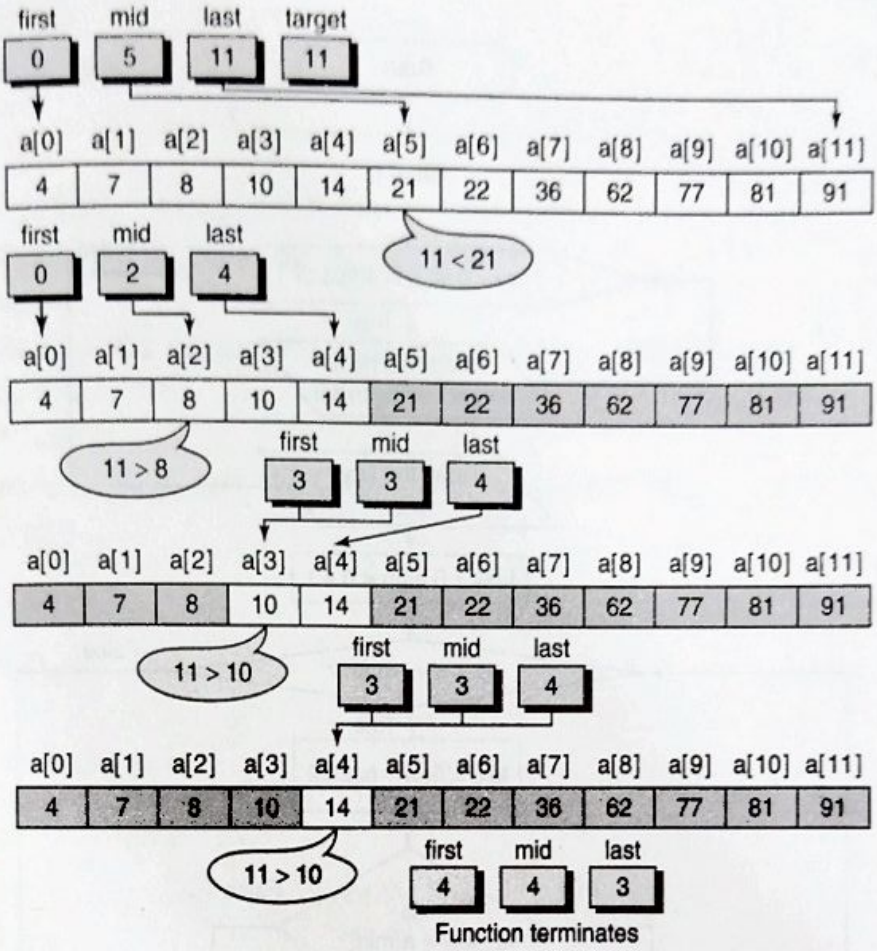
Prerequisites for Coding

Knowledge of loops.

Successful Binary Search Example



Unsuccessful Binary Search Example



Algorithm

Start

Step 1: Read the array size.

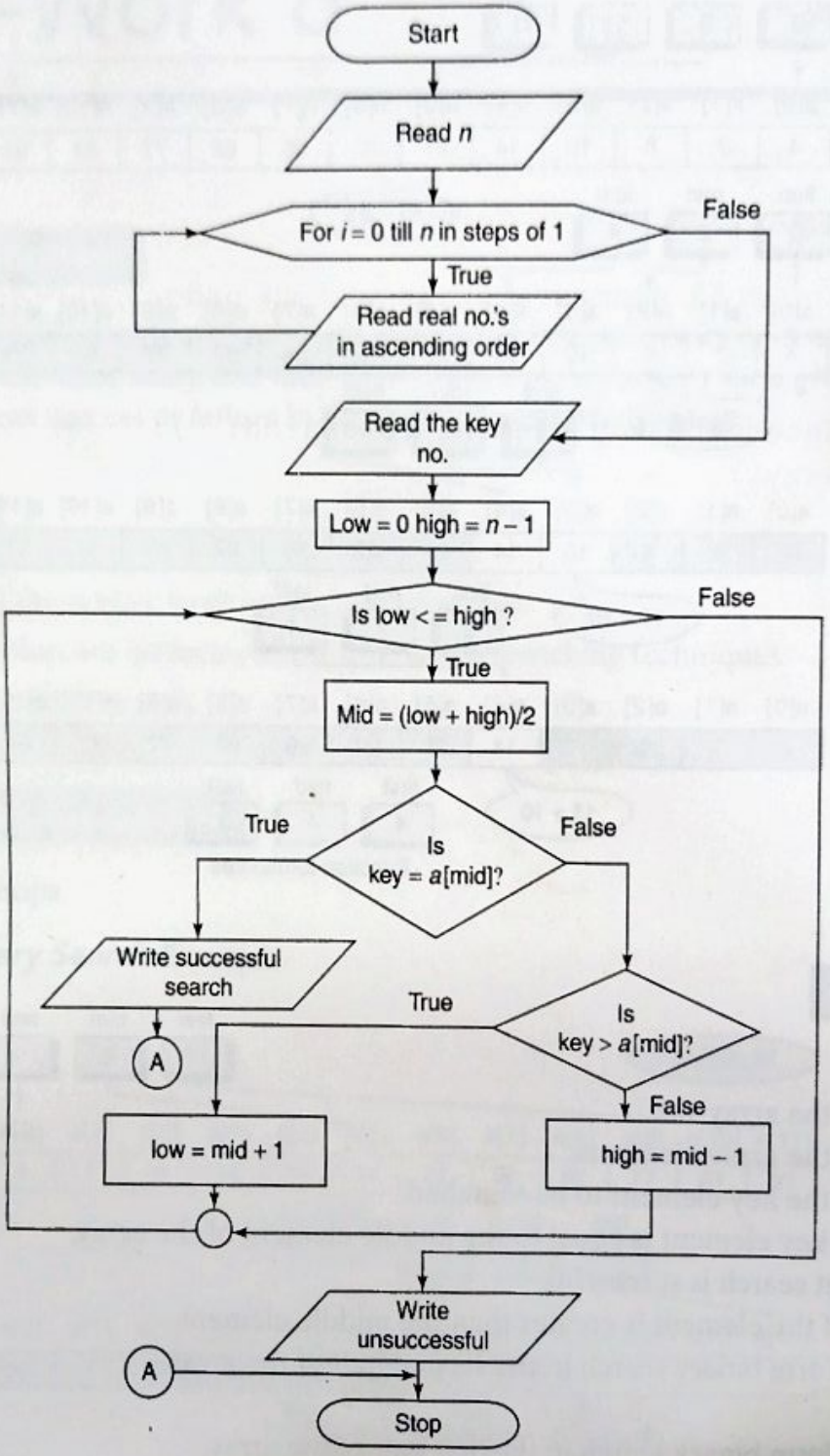
Step 2: Read the array elements.

Step 3: Read the key element to be searched.

Step 4: If the key element is equal to the middle element of the array,
Print search is successful.
Else if the element is greater than the middle element,
Perform binary search in the second half of the array
Else
Perform binary search in the first half of the array.

Stop

Flowchart


Start
Read n
For i = 0 till n in steps of 1
False
True
Read real no.'s in ascending order
Read the key no.
Low = 0 high = n – 1
Is low < = high ?
False
True
Mid = (low + high)/2
True
Is key = a[mid]?
False
Write successful search
A
True
Is key > a[mid]?
False
low = mid + 1
high = mid – 1
Write unsuccessful
A
Stop

Program

```
#include<stdio.h>
#include<process.h>
void main()
{
 int n,i,a[10],key,low,high,mid;
 printf("\n Enter the no. of elements:");
 scanf("%d",&n);
 printf("\n Enter %d elements in ascending order",n);
 for(i=0 ; i<n ; i++)
   scanf("%d",&a[i]);
 printf("\n Enter the key element to search: ");
 scanf("%d",&key);
 low=0;
 high=n-1;
 while(low<=high)
 {
    mid=(low+high)/2;
    if(key==a[mid])
    {
       printf("\nSearch Successful");
       exit(0);
    }
    else if(key>a[mid])
      low=mid+1;
    else
      high=mid-1;
 }
 printf("\n Unsuccessful Search");
}
```

Results

Sample Output 1

```
Enter the no. of elements: 5
Enter 5 elements in ascending order
2
3
4
5
6
```

```
Enter the key element to search: 5
Search Successful
```

Sample Output 2

```
Enter the no. of elements: 8
Enter 8 elements in ascending order
1
12
23
34
56
63
64
88
Enter the key element to search: 66
Unsuccessful Search
```

Sample Output 3

```
Enter the no. of elements: 6
Enter 6 elements in ascending order
11234
12234
23441
23453
26090
30000
Enter the key element to search: 26090
Search Successful
```

Outcomes:

- Should be able to understand storing and retrieving of numbers in an array.
- Able to identify the applications that use binary search such as recursive programs, number guessing games, etc.

Related Program

Program 1

Write a program in C to input N integer numbers in ascending order into a single-dimensional array, and then to perform a binary search for a given key integer number and report success or failure in the form of a suitable message, ***using recursive function.***

```
#include <stdio.h>
#define MAX_LEN 10

/* Recursive function*/
int bsearch_recursive(int l[],int arrayStart,int arrayEnd,int a)
{
 int m,pos;
 if(arrayStart<=arrayEnd)
 {
  m=(arrayStart+arrayEnd)/2;
  if (l[m]==a)
    return m;
  else if (a<l[m])
    return b_search_recursive(l,arrayStart,m-1,a);
  else
    return b_search_recursive(l,m+1,arrayEnd,a);
 }
 return -1;
}

void read_list(int l[],int n)
{
 int i;
 printf("\nEnter the elements:\n");
 for(i=0;i<n;i++)
   scanf("%d",&l[i]);
}

void print_list(int l[],int n)
{
  int i;
  for(i=0;i<n;i++)
    printf("%d\t",l[i]);
 }
 /*main function*/

 void main()
 {
   int l[MAX_LEN], num, ele,f,ll,a;
   int ch,pos;
```

```
  printf("\n Binary Search using Recursion method");
  printf("\n==========================================");
  printf("\nEnter the number of elements: ");
  scanf("%d",&num);
  read_list(l,num);
  printf("\nElements present in the list are:\n\n");
  print_list(l,num);
  printf("\n\nEnter the element you want to search:\n\n");
  scanf("%d",&ele);
  pos=bsearch_recursive(l,0,num,ele);
  if(pos==-1)
  {
    printf("Element is not found");
  }
  else
  {
    printf("Element is found at %d position",pos);
  }
}
```

Output

```
1.
Binary Search using Recursion method
==========================================
Enter the number of elements: 5
Elements present in the list are:
1
2
3
4
5
Enter the element you want to search: 5
Element is found at 4 position

2.
Binary Search using Recursion method
==========================================
Enter the number of elements: 8
Elements present in the list are:
10
```

```
20
30
40
50
60
70
80
Enter the element you want to search: 100
Element is not found
```

Term-Work 7

Problem Definition

Design, develop and execute a program in C to input N integer numbers into a single-dimensional array, sort them in ascending order using bubble sort technique and print both the given array and the sorted array with suitable headings.

Objectives

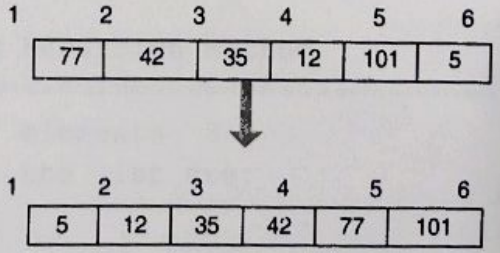
- To accept an input of random set of numbers and store them in an array.
- To understand the logic of bubble sort technique and code it.

Related Theory: Logic Used

Sorting method:

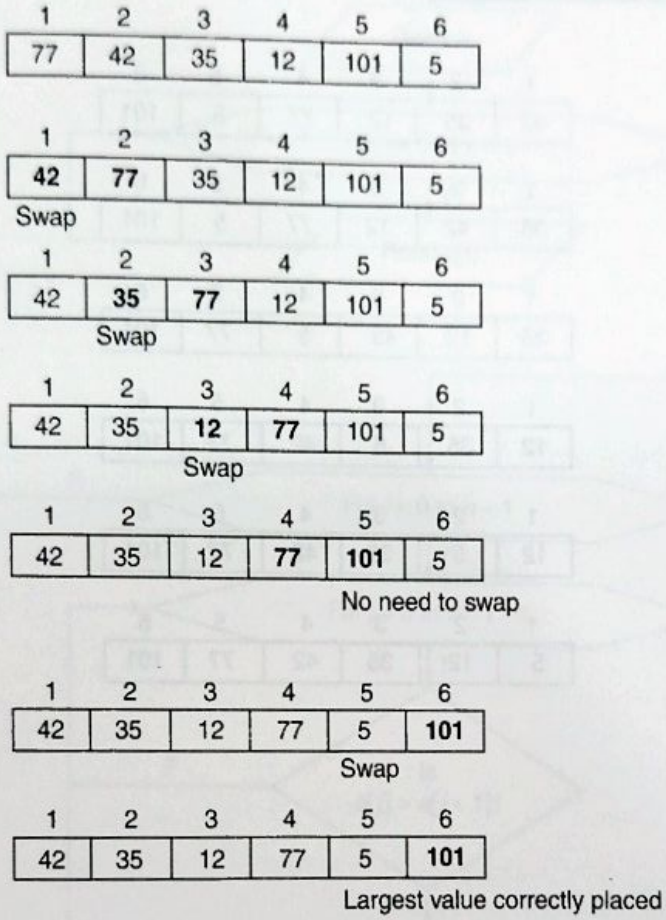
- Bubble sort, also known as sinking sort, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order.
- The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.
- The algorithm gets its name from the way smaller elements "bubble" to the top of the list. It only uses comparisons to operate on elements, hence it is a comparison sort.

Sort: Transforms an Unordered Collection to an Ordered One



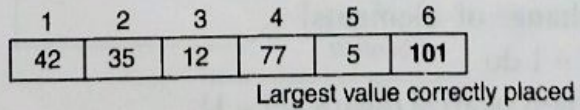
Why is sorting important?

- Sorting enables efficient search.
- For example: Phone Book, dictionary.
- It is important if data has to be searched many times.



Items of Interest

- Notice that only the largest value is correctly placed.
- All other values are still out of order.
- So we need to repeat this process for the unsorted elements.



Repeat "Bubble Up" How Many Times?

- If there are N elements.
- Each time an element is bubbled to its correct location.

- The "bubble up" process is repeated maximum $N-1$ times.
- This guarantees that all the N elements are correctly placed.

"Bubbling" All the Elements

1	2	3	4	5	6
42	35	12	77	5	101

1	2	3	4	5	6
35	42	12	77	5	101

1	2	3	4	5	6
35	12	42	5	77	101

1	2	3	4	5	6
12	35	5	42	77	101

1	2	3	4	5	6
12	5	35	42	77	101

1	2	3	4	5	6
5	12	35	42	77	101

Algorithm

Start

Step 1: [Read the size of an array]
Read n

Step 2: [Read the contents of array elements]
for $i = 0$ to $n-1$
Read $a[i]$

Step 3: [Loop for the number of passes]
for $i = 0$ to $n-1$ do

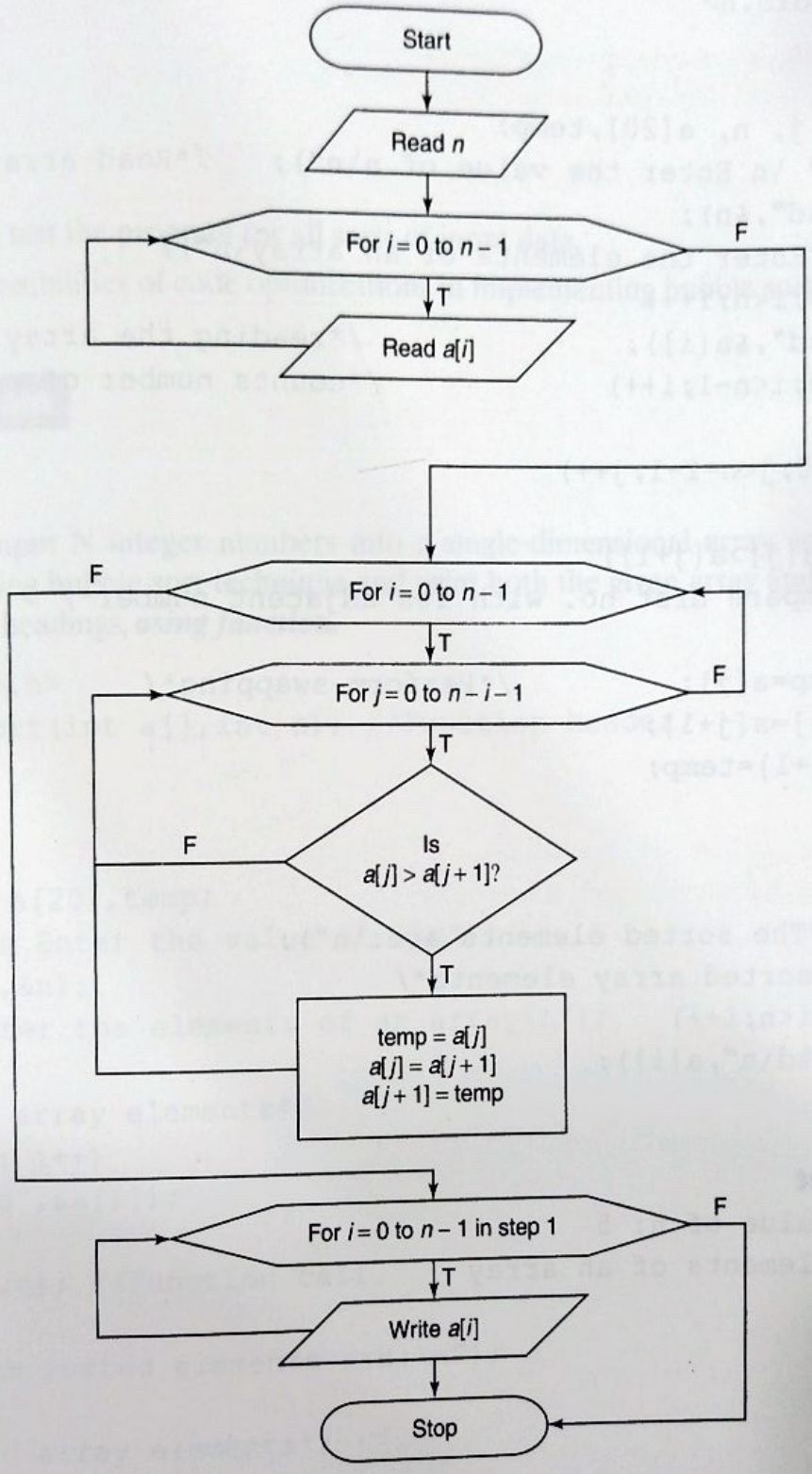
Step 4: [Loop for the exchange of elements]
for $j = 0$ to $n-i-1$ do

Step 5: If $a[j] > a[j+1]$ then swap $a[j]$ with $a[j+1]$
end for loop
end for loop

Step 6: [Print the elements after sorting]
for $i = 0$ to $n-1$
Print $a[i]$

Stop

Flowchart


Start
Read n
For i = 0 to n – 1
F
T
Read a[i]
F
For i = 0 to n – 1
T
For j = 0 to n – i – 1
F
T
F
Is
a[j] > a[j + 1]?
T
temp = a[j]
a[j] = a[j + 1]
a[j + 1] = temp
For i = 0 to n – 1 in step 1
F
T
Write a[i]
Stop

Program

```
#include<stdio.h>
void main()
{
   int  i, j, n, a[20],temp;
   printf(" \n Enter the value of n\n");    /*Read array size*/
   scanf("%d",&n);
   printf("Enter the elements of an array\n");
   for (i=0;i<n;i++)
   scanf("%d",&a[i]);                      /*reading the array elements*/
   for (i=0;i<n-1;i++)                    /*counts number of passes*/
   {
     for(j=0;j<n-i-1;j++)
     {
       if (a[j]>a[j+1])
       /*Compare first no. with its adjacent number*/
       {
         temp=a[j];              /*Perform swapping*/
         a[j]=a[j+1];
         a[j+1]=temp;
       }
     }
   }
   printf("The sorted elements are:\n");
   /*Print sorted array elements*/
   for(i=0;i<n;i++)
   printf("%d\n",a[i]);
}
```

Sample Output

```
Enter the value of n: 5
Enter the elements of an array
3
6
1
0
9

The sorted elements are
0
```

```
1
3
6
9
```

Outcomes

- Should be able to test the program for all sorts of input data.
- To find out the possibilities of code optimizations in implementing bubble sort technique.

Related Program

Program 1

Program in C to input N integer numbers into a single-dimensional array, sort them in ascending order using bubble sort technique and print both the given array and the sorted array with suitable headings, ***using function.***

```
#include<stdio.h>
void Bubble_sort(int a[],int n); //Function header.

void main()
{
int  i, j, n, a[20],temp;
   printf(" \n Enter the value of n\n");
   scanf("%d",&n);
   printf("Enter the elements of an array\n");

/*reading the array elements*/
for (i=0; i<n; i++)
    scanf("%d",&a[i]);

Bubble_sort(a,n); //Function call.

   printf("The sorted elements are:\n");

/*Print sorted array elements*/
   for(i=0;i<n;i++)
   printf("%d\n",a[i]);
}
```

```
void Bubble_sort(int a[],int n) //Function defination.
{
  int i,j,temp;
  /*counts number of passes*/
  for (i=0;i<n-1;i++)
  {
    for(j=0;j<n-i-1;j++)
    {
      /*Compare first no. with its adjacent number*/
        if (a[j]>a[j+1])
          /*Perform swapping*/
          temp=a[j];
    a[j]=a[j+1];
      a[j+1]=temp;
     }
    }
   }
}
```

Output

```
1.
Enter the value of n: 5
Enter the elements of an array
3
6
1
0
9
The Sorted elements are
0
1
3
6
9

2.
Enter the value of n: 7
Enter the elements of an array
30
62
```

```
11
45
78
21
24
The Sorted elements are
11
21
24
30
45
62
78
```

Term-Work 8

Problem Definition

Design, develop and execute a program in C to compute and print the word length on the host machine.

Objective

To compute the number of bits used to store a given integer.

Prerequisites for Coding

Do While Loop Syntax

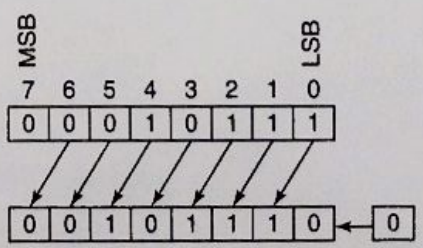
```
Initialization statement
  do
  {
    executable statements;
    increment/decrement statement;
  } while(condition);
```

Knowledge of Loops and Bitwise Operators

Shift operators →

1. Left shift (<<)
2. Right shift (>>)

Syntax



int $a = 23, b = 1$;
$a = 23$ is represented in binary form as

0001 0111

$a << b \rightarrow$ is to shift a left b times

After executing this statement,

$a = 0010\ 1110$ (46 – decimal value)

Complement Operator (~)

Bitwise complement operator

Complements each bit.

For example, int $n = 0$; /* will store binary zero in memory location*/

If int data type occupies 32 bits then,

$n = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$

$n = \sim n$; /*will store all 1's in memory location*/

Then $\sim n = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111$

Related Theory

To find word length, we need to know how many bits are actually used to store a given integer number. For this, the following logic is employed.

Program Logic

- **Step 1 :** Store 1 in all bit positions of an integer variable. This is done by complimenting the number 0 and assigning the result to a program variable $N = \sim 0$;

1	1	1	1	1	1	1	1

This will store all 1's in all the bit positions of the number.

- **Step 2 :** Perform repeated Shift left operation on the number and check if it has become zero and every time you shift, increment a counter.

Logic to Find Word Length of a Host Machine

Using given logic,

```
n=~0; count=0;
do
{
  n=n<<1;
  count++;
}while(n!=0);
```

Algorithm

Start

Step 1: [Initialize *n* to complement of zero and count to zero]

word = ~0 and count = 0

Step 2: [Count no of 1's in *n*]

Repeat

word = word << 1

increment count

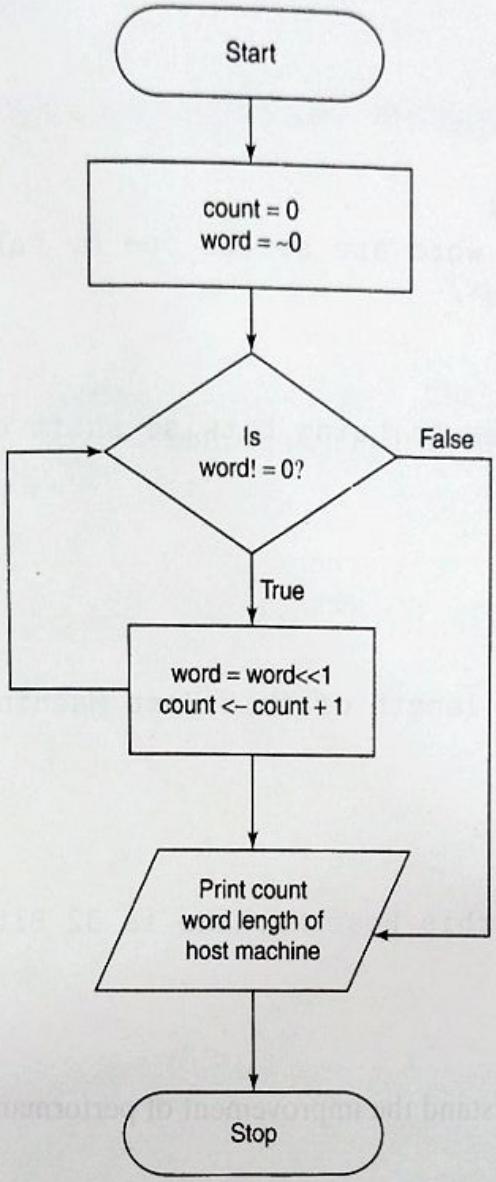
until word = 0

Step 3: [Print count]

Print count as word length of host machine

Stop

Flowchart


Start
count = 0
word = ~0
Is
word! = 0?
False
True
word = word<<1
count <- count + 1
Print count
word length of
host machine
Stop

Program

```
#include<stdio.h>
void main()
{
  int count=0;
  int unsigned word;
  /*All bits of the word are set to one by taking one's
  complement of zero*/
  word=~0;

  /*Counting Word Length using bitwise shift operator*/
  while(word!=0)
  {
    word=word<<1;
    ++count;
  }
  printf("\The word length of this Host Machine is %d Bits
  \n",count);
}
```

Sample Output

```
The word length of this Host Machine is 32 Bits
```

Outcomes

- Should be able to understand the improvement of performance of CPU with increase in word length.
- Should be able to identify the machines whose word lengths are 16, 32, 64, etc.

Related Program

Program 1

Write a program in C to compute and print the word length on the host machine, ***using do... while loop***.

```
#include<stdio.h>
void main()
{
  int count=0;
```

```
int unsigned word;
/*All bits of the word are set to one by taking one's
complement of zero*/
word=~0;
/*Counting Word Length using bitwise shift operator*/
do
{
  word=word<<1;
  ++count;
}while(word!=0);
printf("\n The word length of this Host Machine is %d Bits
\n",count);
}
```

Output

```
The word length of this Host Machine is 32 Bits
```

Term-Work 9

Problem Definition

Design, develop and execute a program in C to calculate the approximate value of exp(0.5) using the Taylor series expansion for the exponential function. Use the terms in the expansion until the last term is less than the machine epsilon defined as FLT_EPSILON in the header file <float.h>. Print the value returned by the mathematical function exp( ).

Objectives

- To understand Taylor series and implement the same using C program.
- To identify standard built-in library functions and corresponding header files to be included.
- To compare the computed result with the value returned by standard built-in functions for accuracy.

Related Theory

Taylor series is a way of expressing a function as a sum of simple but infinite number of terms. For example, suppose you want to find exp(3.654), Taylor series expansion allows you to find this value as a sum of simple terms. The Taylor series form of exp(x) is

$$\exp(x) = 1 + (x/1!) + (x\text{^}2/2!) + (x\text{^}3/3!) + \cdots$$

(! means factorial). If you know the value of x, you can calculate the terms very easily and add them up to required accuracy. In statistics and mathematics, Taylor series is used for simplification and approximation.

- The terms in the expansion should be added until the last term is less than the machine epsilon

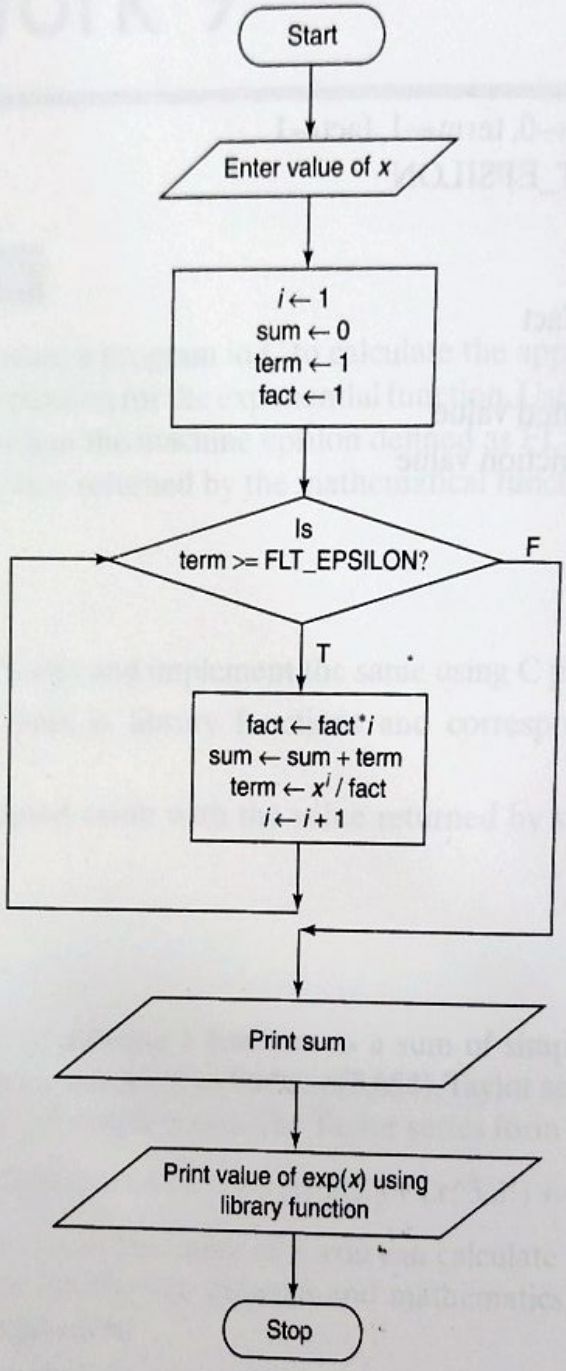
  Represented as FLT_EPSILON
- FLT_EPSILON macro is defined in float.h and should be included in C program as

  #include<float.h>.
- FLT_EPSILON is the minimum positive floating point number of type float. It's supposed to be no greater than 1E-5.
- The exp function returns the value of e (the base of natural logarithms) raised to power x.

Algorithm

Start
Step 1: Input x
Step 2: Initialize i←1, sum←0, term←1, fact←1
Step 3: While term >= FLT_EPSILON
fact <– fact*i
sum <– sum+term
term <– pow(x, i)/fact
Increment i by 1
Step 4: Output the calculated value
Step 5: Output Library function value
Stop

Flowchart


Start
Enter value of x
i ← 1
sum ← 0
term ← 1
fact ← 1
Is
term >= FLT_EPSILON?
F
T
fact ← fact * i
sum ← sum + term
term ← x^i / fact
i ← i + 1
Print sum
Print value of exp(x) using
library function
Stop

Program

```
#include<stdio.h>
#include<float.h>
#include<math.h>
void main()
{
 int i;
 float x,sum,fact,term;
 printf("\n You have this series : 1+x/1!+x^2/2!+x^3/3!+
 X^4/4!+_ _ _ _ _ _x^n/n!");
 printf("\n\n Enter the value for X:");
 scanf("%f",&x);
 i=1;
 sum=0;
 term=1;
 fact=1;
 while(term>=FLT_EPSILON)
 {
   fact*=i;
   sum+=term;
   term=pow(x,i)/fact;
   i++;
 }
 printf("\n\n The Calculated value of e^% .3f = %f",x,sum);
 printf("\n\n The Library Function Value of e^%.3f = %f",x,
 exp(x));
}
```

Sample Output

```
You have this series: 1+x/1!+x^2/2!+x^3/3!+x^4/4!+.....x^n/n!
Enter the value for X: 0.5
The Calculated value of e^0.500=1.648721
The Library Function Value of e^0.500=1.648721
```

Outcomes

- Should be able to write C programs for any **statistical and mathematical** functions.
- Capable of refining the program till the required accuracy is attained.

Practice Programs

1. Write a C program to find the sum of sine series $1 + x\text{^}3/3! + x\text{^}5/5! + \cdots$.
2. Write a C program to find the sum of cosine series $x\text{^}2/2! + x\text{^}4/4! + \cdots$.

Term-Work 10

Problem Definition

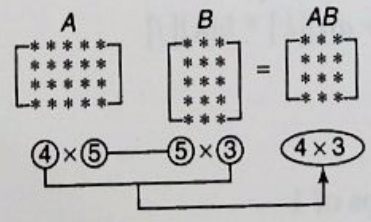
Design, develop and execute a program in C to read two matrices $A(M*N)$ and $B(P*Q)$ and to compute the product of A and B if the matrices are compatible for multiplication. The program is to print the input matrices and resultant matrix with suitable headings and format if the matrices are compatible for multiplication. Otherwise the program must print a suitable message. (For the purpose of demonstration, the array sizes M, N, P and Q can all be less than or equal to 3.)

Objectives

- To read data and store in two-dimensional arrays (matrix form).
- To identify the compatibility of matrices for multiplication and print suitable messages.
- To write a C program to perform matrix multiplication operation on arrays.

Related Theory

Multiplication of two matrices, i.e., A $(m \times n)$ and B $(p \times q)$ should check a condition, i.e., number of columns in A = number of rows in B. If this condition satisfies, then multiplication is possible otherwise not. Hence, the resultant matrix will be C $(m \times q)$. If A is a 4×5 matrix and B is a 5×3 matrix, what are the sizes of AB, if they are defined?



Consider this A (2×2) and B (2×2) matrix multiplication and the result is in C (2×2).

$$C_{00} = A_{00}*B_{00} + A_{01}*B_{10}$$
$$C_{01} = A_{01}*B_{11} + A_{00}*B_{01}$$
$$C_{11} = A_{11}*B_{01} + A_{10}*B_{01}$$
$$C_{10} = A_{10}*B_{00} + A_{11}*B_{10}$$

$$A = \begin{matrix} 3 & 0 \\ 1 & 1 \end{matrix} \quad (2 \times 2) \qquad B = \begin{matrix} 4 & 7 \\ 6 & 8 \end{matrix} \quad (2 \times 2)$$

$$A \times B =$$

$$3(4) + 0(6) = 12$$
$$3(7) + 0(8) = 21$$
$$1(4) + 1(6) = 10$$
$$1(7) + 1(8) = 15$$

Algorithm

Start

Step 1: Input order m, n

Step 2: FOR $i \leftarrow 0$ to $m - 1$ in steps of 1
  FOR $j \leftarrow 0$ to $n - 1$ in steps of 1
    Input $a[i][j]$
  END FOR
END FOR

Step 3: Input order p, q

Step 4: FOR $i \leftarrow 0$ to $p - 1$ in steps of 1
  FOR $j \leftarrow 0$ to $q - 1$ in steps of 1
    Input $b[i][j]$
  END FOR
END FOR

Step 5: IF $n = p$ THEN
FOR $i \leftarrow 0$ to $m - 1$ in steps of 1
  FOR $j \leftarrow 0$ to $q - 1$ in steps of 1
    $c[i][j] = 0$
      FOR $k \leftarrow 0$ to $n - 1$ in steps of 1
        $c[i][j] = c[i][j] + a[i][k] * b[k][j]$
      END FOR
  END FOR
END FOR

Step 6: FOR $i \leftarrow 0$ to $m - 1$ in steps of 1
  FOR $j \leftarrow 0$ to $n - 1$ in steps of 1
    output $a[i][j]$
  END FOR
END FOR

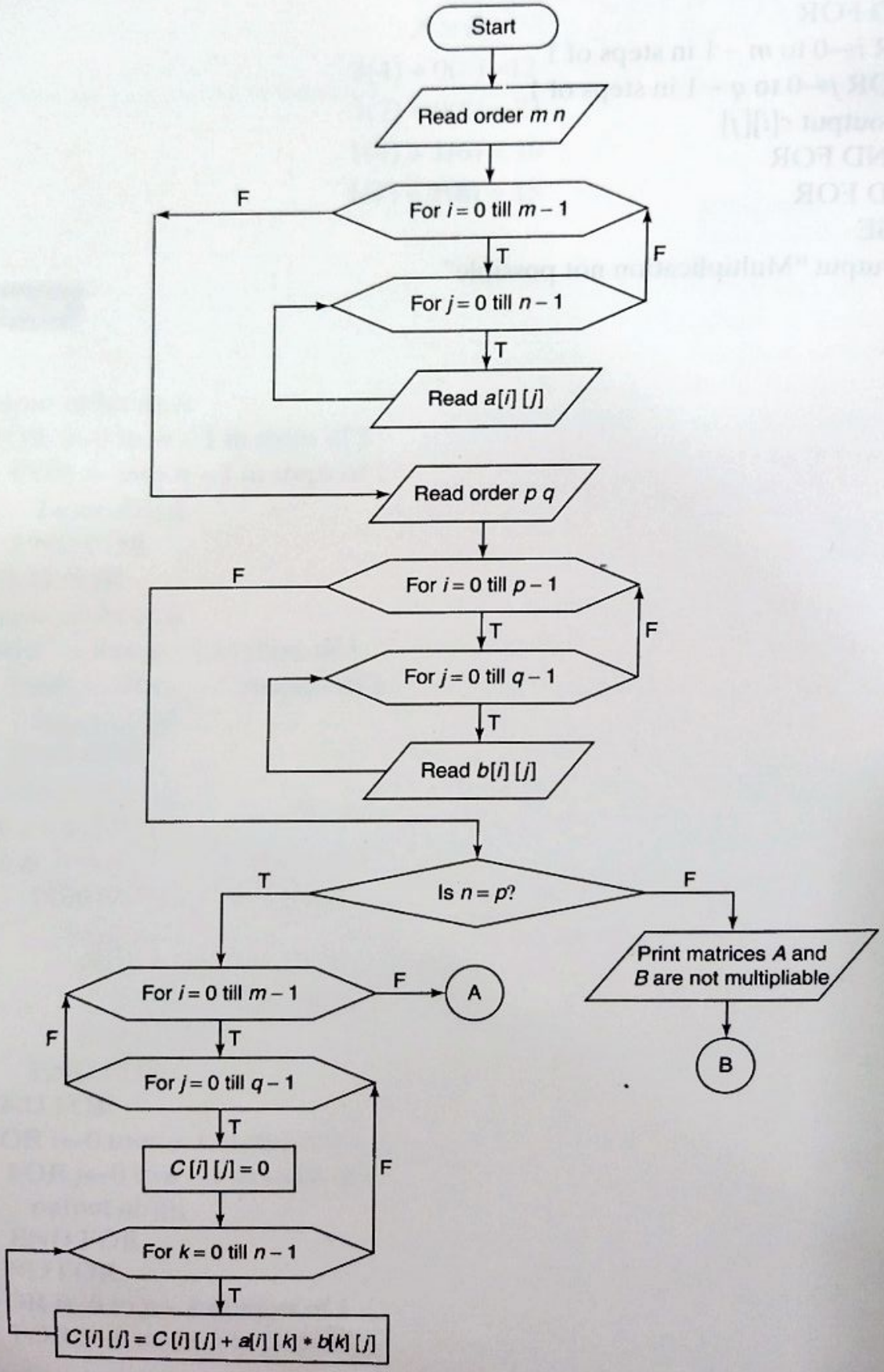
Step 7: FOR $i \leftarrow 0$ to $p - 1$ in steps of 1
  FOR $j \leftarrow 0$ to $q - 1$ in steps of 1

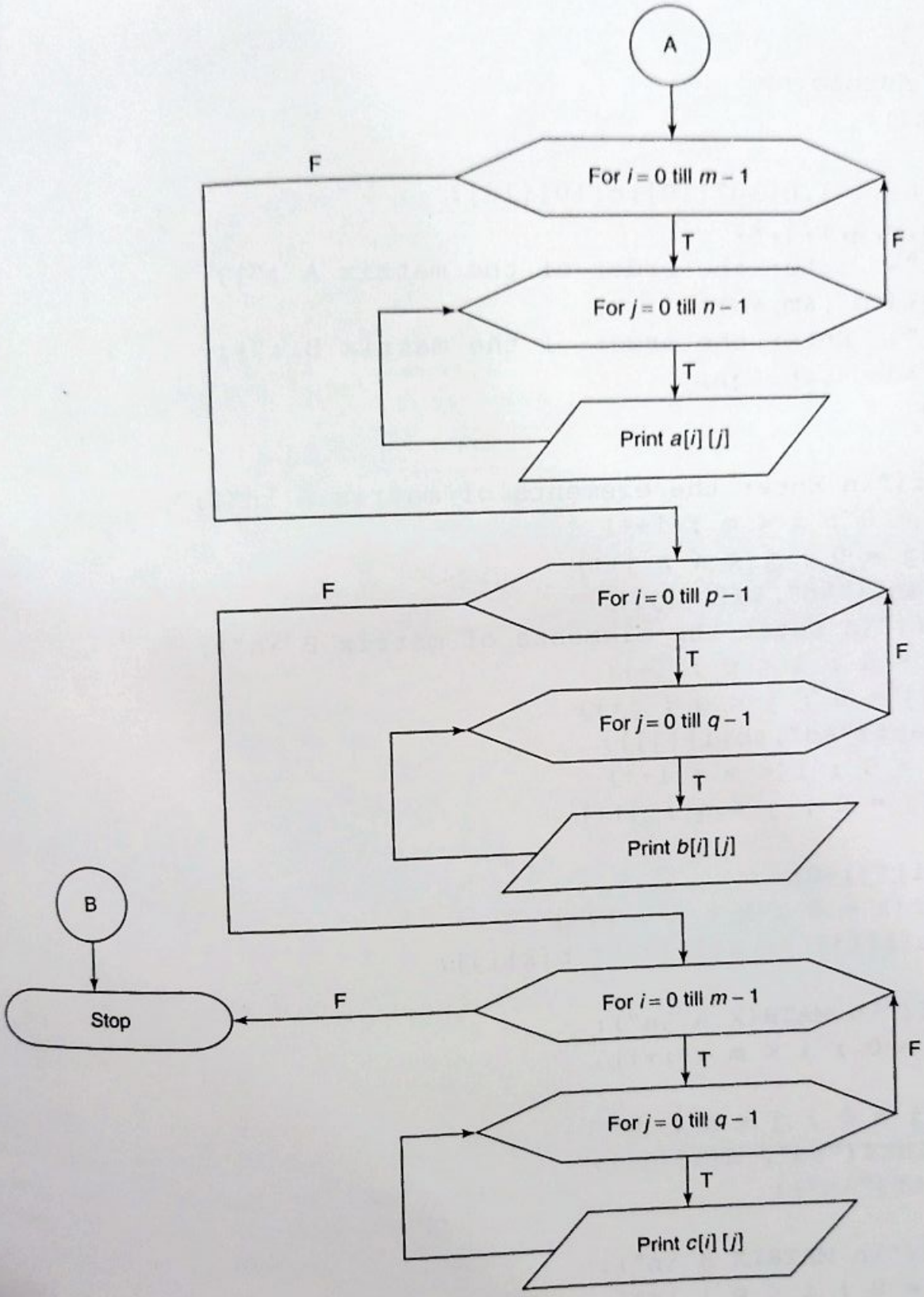
output $b[i][j]$
END FOR
END FOR

Step 8: FOR $i \leftarrow 0$ to $m - 1$ in steps of 1
FOR $j \leftarrow 0$ to $q - 1$ in steps of 1
output $c[i][j]$
END FOR
END FOR
ELSE
Output "Multiplication not possible"

Stop

Flowchart


Start
Read order m n
For i = 0 till m – 1
F
T
F
For j = 0 till n – 1
T
Read a[i] [j]
Read order p q
For i = 0 till p – 1
F
T
F
For j = 0 till q – 1
T
Read b[i] [j]
Is n = p?
T
F
Print matrices A and B are not multipliable
B
For i = 0 till m – 1
F
A
F
T
For j = 0 till q – 1
T
C [i] [j] = 0
F
For k = 0 till n – 1
T
C [i] [j] = C [i] [j] + a[i] [k] * b[k] [j]


A
For $i = 0$ till $m - 1$
F
T
F
For $j = 0$ till $n - 1$
T
Print $a[i]$ $[j]$
For $i = 0$ till $p - 1$
F
T
F
For $j = 0$ till $q - 1$
T
Print $b[i]$ $[j]$
B
Stop
F
For $i = 0$ till $m - 1$
T
F
For $j = 0$ till $q - 1$
T
Print $c[i]$ $[j]$

Program

```
#include <stdio.h>
void main()
{
  int a[10][10],b[10][10],c[10][10];
  int m,n,p,q,i,j,k;
  printf("\n Enter the order of the matrix A :");
  scanf("%d%d",&m,&n);
  printf("\n Enter the order of the matrix B :");
  scanf("%d%d",&p,&q);
if (n==p)
  {
    printf("\n Enter the elements of matrix A \n");
    for(i = 0 ; i < m ; i++)
      for(j = 0 ; j < n ; j++)
        scanf("%d",&a[i][j]);
    printf("\n Enter the elements of matrix B \n");
    for(i = 0 ; i < p ; i++)
      for(j = 0 ; j < q ; j++)
        scanf("%d",&b[i][j]);
    for(i = 0 ; i < m ; i++)
      for(j = 0 ; j < q ; j++)
      {
        c[i][j]=0;
        for(k = 0 ; k < n ; k++)
          c[i][j] += a[i][k] * b[k][j];
      }
    printf("\n MATRIX A \n");
    for(i = 0 ; i < m ; i++)
    {
      for(j = 0 ; j < n ; j++)
        printf("%d", a[i][j]);
      printf("\n");
    }
    printf("\n MATRIX B \n");
    for(i = 0 ; i < p ; i++)
    {
      for(j = 0 ; j < q ; j++)
        printf("%d", b[i][j]);
      printf("\n");
```

```
  }
  printf("\n MATRIX C \n");
  for(i = 0 ; i < m ; i++)
  {
    for(j = 0 ; j < q ; j++)
      printf(" %d ", c[i][j]);
    printf("\n");
  }
 }
 else
   printf("\nMatrix A & B is not multiplicable");
}
```

Sample Output

```
Enter the order of the matrix A: 2 2
Enter the order of the matrix B: 2 2
Enter the elements of matrix A
1  2
3  4
Enter the elements of matrix B
2  3
4  5
MATRIX A
1  2
3  4
MATRIX B
2  3
4  5
MATRIX C
10   13
22   29
--------------------------------------------------------------------
Enter the order of the matrix A: 3 2
Enter the order of the matrix B: 3 4
Matrix A & B is not multipliable
```

Outcomes

- Able to represent the data in matrix form.
- Capable of writing code to perform various operations on matrices.

Related Programs

Program 1

Write a C program to find the sum of the principal diagonal elements of a 3*3 matrix.

```
#include<stdio.h>
void main()
{
  int a[3][3],sum,i,j;
  printf("\n Enter 9 elements of matrix\n");
  for(i=0;i<3;i++)
  for(j=0;j<3;j++)
    scanf("%d",&a[i][j]);
  /*printing in matrix form*/
  printf("Given input matrix is...\n");
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
      printf("%d",a[i][j]);
    printf("\n");
  }
  /*to find sum of principal diagonal elements*/
  sum=0;
  for(i=0;i<3;i++)
  for(j=0;j<3;j++)
  if(i==j)
    sum=sum+a[i][j];
  printf("sum of principal diagonal elements =%d\n",sum);
}
```

Program 2

Write a C program to find the sum of two matrixes A and B of order $m \times n$.

```
#include<stdio.h>
void main()
{
  int m,n,a[10][10],i,j,b[10][10],c[10][10];
  printf("what is order of matrices a & b \n");
  scanf("%d%d",&m,&n);
  printf("enter %d elements of matrix a \n",m*n);
  for(i=0;i<m;i++)
```

```
for(j=0;j<n;j++)
  scanf("%d",&a[i][j]);
printf("given input in matrix form of a \n");
for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  printf("%d",a[i][j]);
  printf("\n");
}
printf("enter %d elements of matrix b \n",m*n);
for(i=0;i<m;i++)
for(j=0;j<n;j++)
  scanf("%d",&b[i][j]);
printf("given input in matrix form of b \n");
for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  printf("%d",b[i][j]);
  printf("\n");
}
for(i=0;i<m;i++)
for(j=0;j<n;j++)
c[i][j]=a[i][j]+b[i][j]; /*addition */
  printf("sum matrix c is....\n");
for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  printf("%d",c[i][j]);
  printf("\n");
 }
}
```

Term-Work 11

Problem Definition

Design, develop and execute a parallel program in C to add, element-wise, two one-dimensional arrays *A* and *B* of *N* integer elements and to store the result in another one-dimensional array *C* of *N* integer elements.

Objectives

- To perform addition of two one-dimensional arrays in parallel.
- To understand the concept and usage of thread basics and OpenMP directives.

Prerequisites for Coding

- The basic requirement to write this program is to have the knowledge of parallel computing
- *Thread basics, OpenMP(Open MultiProcessing), array and looping constructs.*

Related Theory: Logic Used

Difference between Sequential and Parallel Programming

Sequential Programming	*Parallel Programming*
Sequential programming involves a consecutive and ordered execution of processes one after another	Parallel programming involves the concurrent computation or simultaneous execution of processes or threads
Sequential programming, processes are run one after another in a succession fashion	Parallel computing, multiple processes execute at the same time

Parallel Computing

Parallel computing is the execution of multiple tasks concurrently in a computer system. For example, a large problem to be solved can be broken into multiple small-sized problems which are then solved in parallel to arrive at the final solution.

Thread

A thread is the smallest processing unit in a system. In parallel programming, several different threads are created corresponding to different computational elements and are then executed in parallel.

OpenMP (Open MultiProcessing)

It is an API that supports directive-based programming. It provides a number of built-in directives and library functions that facilitate the development of parallel computing applications.

In C, an OpenMP directive is specified with the help of **#pragma** directive, as shown below:

#pragma omp directive [clause 1] [clause 2] .....

- ***#pragma omp*** tells the compiler about the specification of an OpenMP directive
- ***directive*** is a valid OpenMP directive name.
- ***clause*** list comprises a set of clauses associated with the directive.

The Parallel Directive

It is the OpenMP directive that helps to specify a block of code to be executed in parallel via multiple threads.

#pragma omp parallel

- For parallel programming in C, we make use of a ***#pragma omp parallel*** expression that is defined in ***"omp.h"*** header file.
- So when the compiler comes across the ***#pragma omp parallel*** then the following block of statements will be executed in parallel as separate threads.

The for Directive

The for directive is specified before a for loop and it specifies that the iterations of the for loop be divided across multiple threads.

#pragma omp parallel for

Algorithm

Start

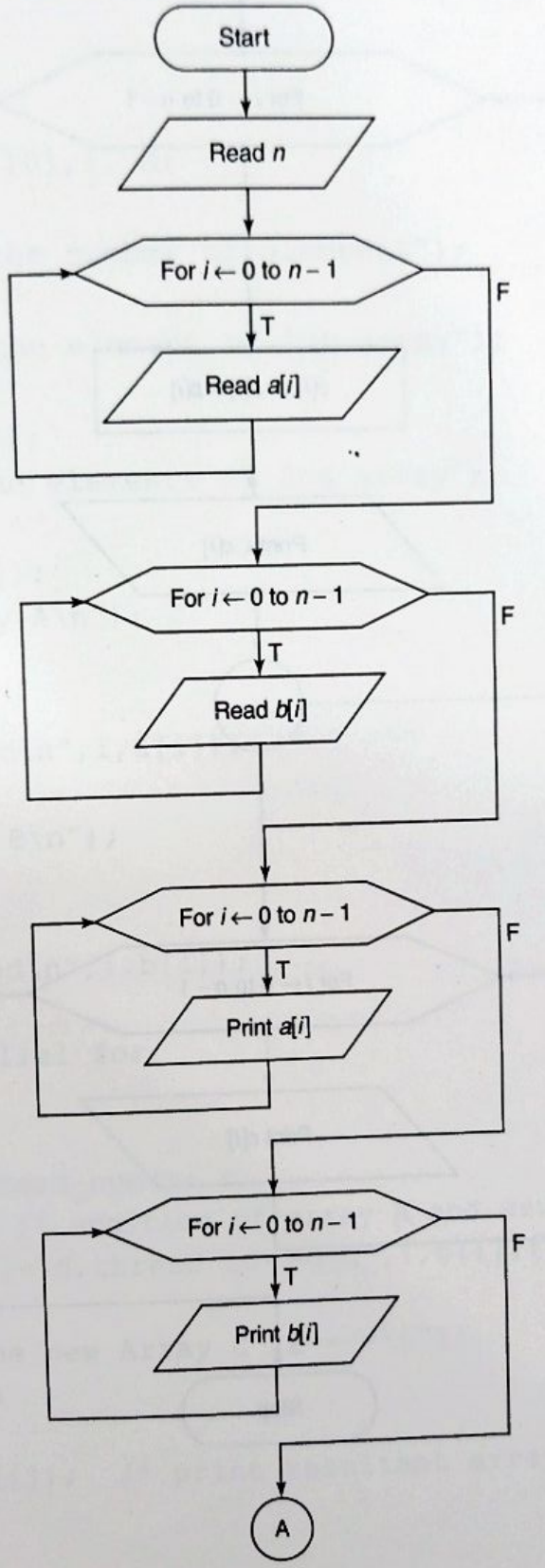
Step 1: Read the number of elements

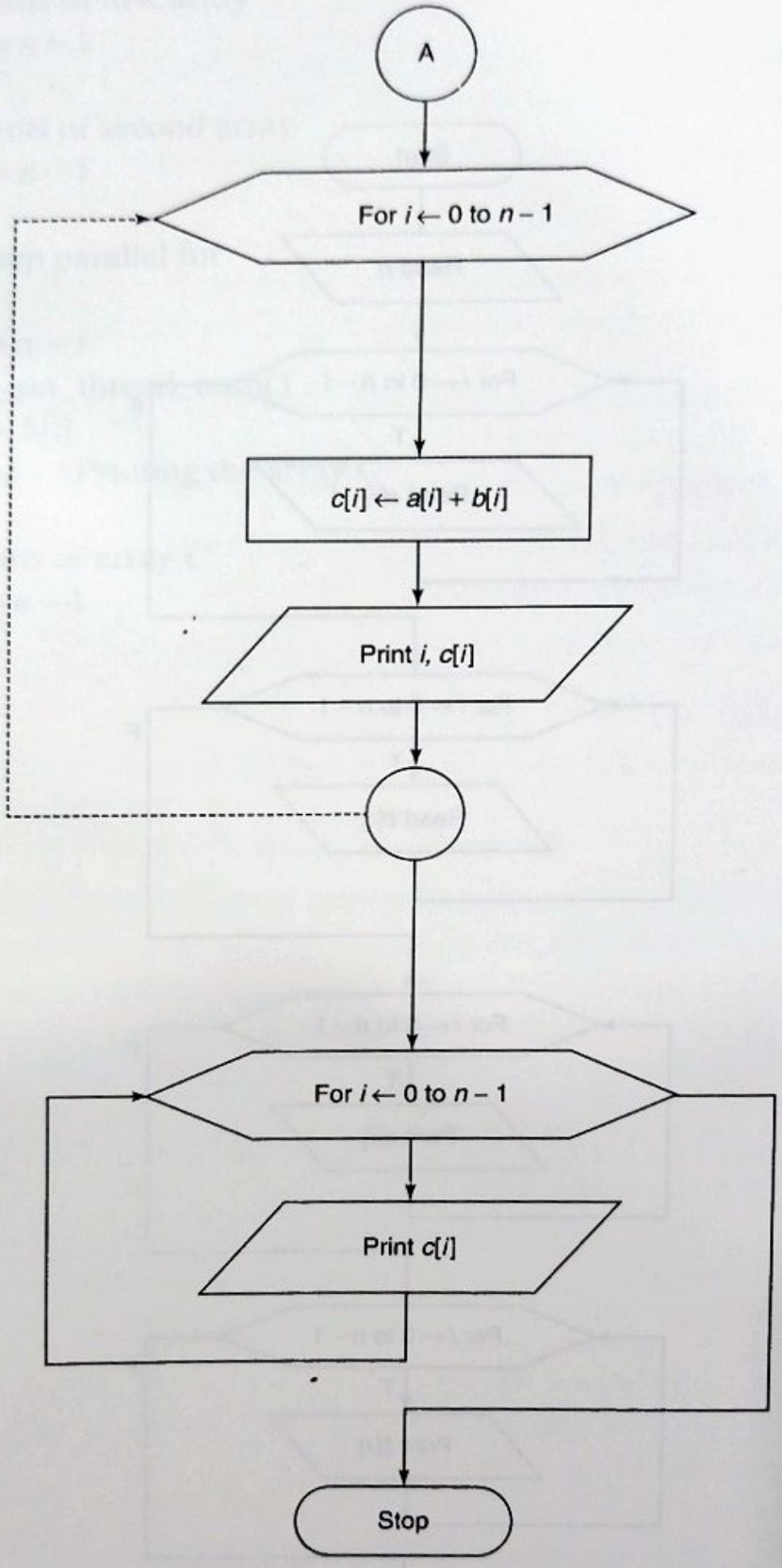
Step 2: Read the elements of first array
for $i \leftarrow 0$ to $n-1$
Read $a[i]$

Step 3: Read the elements of second array
for $i \leftarrow 0$ to $n-1$
Read $b[i]$

Step 4: Print contents of first array
for $i \leftarrow 0$ to $n-1$
print $a[i]$
Step 5: Print contents of second array
for $i \leftarrow 0$ to $n-1$
Print $b[i]$
Step 6: #pragma omp parallel for
begin
for $i \leftarrow 0$ to $n-1$
tid $\leftarrow$ omp_get_thread_num()
$c[i] \leftarrow a[i] + b[i]$
print $c[i]$, tid //Printing the array C
end
Step 7: Print contents of array C
for $i \leftarrow 0$ to $n-1$
print $c[i]$
Stop

Flowchart


Start
Read n
For i ← 0 to n – 1
F
T
Read a[i]
For i ← 0 to n – 1
F
T
Read b[i]
For i ← 0 to n – 1
F
T
Print a[i]
For i ← 0 to n – 1
F
T
Print b[i]
A


A
For $i \leftarrow 0$ to $n-1$
$c[i] \leftarrow a[i] + b[i]$
Print i, $c[i]$
For $i \leftarrow 0$ to $n-1$
Print $c[i]$
Stop

Program

```
#include<stdio.h>
#include<omp.h>
void main()
{
 int a[10],b[10],c[10],i, n;
 int tid;
 printf("\n Enter the number of elements");
 scanf("%d",&n);
 printf("\n Enter the element of 1st array");
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
  printf("Enter the elements of 2nd array");
 for(i=0;i<n;i++)
  scanf("%d",&b[i]);
  printf("\n Array A\n");
 for(i=0;i<n;i++)
 {
  printf("a[%d]=%d\n",i,a[i]);
 }
 printf("\n Array B\n");
 for(i=0;i<n;i++)
 {
  printf("b[%d]=%d\n",i,b[i]);
 }
 #pragma omp parallel for
 for(i=0;i<n;i++)
 {
  tid=omp_get_thread_num();
  c[i]=a[i]+b[i]; /* addition of array A and array B in parallel */
  printf("\nc[%d]=%d,thread id=%d\n",i,c[i],tid);
 }
 printf("\n====the new Array C is====\n");
 for(i=0;i<n;i++)
 {
  printf("%d",c[i]);  /* print resultant array C */
 }
}
```

Sample Output 1

```
Enter the number of elements 5
Enter the elements of 1st array  1 2 3 4 5
Enter the elements of 2nd array  5 6 7 8 9

Array A
a[0]=1
a[1]=2
a[2]=3
a[3]=4
a[4]=5

Array B
b[0]=5
b[1]=6
b[2]=7
b[3]=8
b[4]=9

c[0]=6,thread id=0
c[1]=8,thread id=0
c[2]=10,thread id=0
c[3]=12,thread id=1

c[4]=14,thread id=1
====the new Array C is====
6     8     10     12     14

Enter the number of elements 6
Enter the element of 1st array  10 20 30 40 50 60
Enter the elements of 2nd array  1 2 3 4 5 6

Array A
a[0]=10
a[1]=20
a[2]=30
a[3]=40
a[4]=50
a[5]=60
```

```
Array B
b[0]=1
b[1]=2
b[2]=3
b[3]=4
b[4]=5
b[5]=6

c[0]=11,thread id=0
c[1]=22,thread id=0
c[2]=33,thread id=0
c[3]=44,thread id=1
c[4]=55,thread id=1
c[5]=66,thread id=1
====the new Array C is====
11   22   33   44   55   66   77   88
```

Outcomes

- Able to differentiate between sequential and parallel programming.
- Capable of identifying applications where parallelism can be applied.

Practice Programs

1. Write a C program to create multiple threads in a program and print their thread Id's.
2. Write a parallel program to multiply two square matrices.

Term-Work 12

Problem Definition

Design and develop a function rightrot(x, n) in C that returns the value of the integer x rotated to the right by n bit positions as an unsigned integer. Invoke the function from the main with different values for x and n and print the results with suitable headings

Objectives

- To develop a function rightrot to rotate a number toward the right by a given number of bit positions and return the rotated number to main function.
- To understand the concepts of using functions and parameter passing.
- To test the program for all possible input cases.

Prerequisites for Coding

The basic requirement to write this program is to have the knowledge of bitwise operators.

Functions in C

The programs we have presented so far have been very simple. They solved problems that could be understood without too much effort. The principles of top–down design and structured programming dictate that a program should be divided into a main module and its related modules. Each module should also be divided into sub modules according to software engineering principles.

In C, the idea of top–down design is done using functions. A C program is made of one or more functions, one and only one of which must be named main. In general, the purpose of a function is to receive zero or more pieces of data, operate on them, and return at most one piece of data.

A function in C is an independent module that will be called to do a specific task. When it is called it receives zero or more pieces of data, operates on them and returns at most one piece of data to the calling function.

Note: In C, a program is made of one or more functions, one and only one of which must be called *main*.

The execution of the program always starts with main, but it can call other functions to do some part of the job.

User-Defined Functions

Like every other object in C, functions must be both declared and defined. The function declaration gives the whole picture of the function that needs to be defined later. The function definition contains the code for a function.

A function name is used three times: for declaration, in a call, and for definition.

Function Design Strategies

Basically, there are four function design strategies, namely

1. void function without parameters.
2. void function with parameters.
3. non-void function without parameters.
4. non-void function with parameters.

Sample Program

Program in C to find the GCD of two integers and to output the result, ***using function***.

```
#include<stdio.h>
#include<process.h>
int GCD(int a, int b)
{
 if (a<0) a = -a;
 if (b<0) b = -b;
 if (b>a)
 {
  printf("\n Invalid Input");
  exit(0);
 }
 else
 {
  if (a==0 || b==1 || a==b) return b;
  if (a==1 || b==0) return a;
  if (a>b) return GCD(b, a%b);
  else return GCD(a, b%a);
 }
}
```

```
void main()
{
  int x,y;
  printf("\n Enter 1st number:");
  scanf("%d",&x);
  printf("\n Enter 2nd number:");
  scanf("%d",&y);
  printf("\n GCD is:%d", GCD(x,y));
}
```

For three numbers

```
int GCD3 (int x, int y, int z)
{
  return GCD(x, GCD(y, z));
}
```

Output

```
Enter 1st number: 10
Enter 2nd number: 5
GCD is: 5

Enter 1st number: 20
Enter 2nd number: 15
GCD is: 5

Enter 1st number: 8
Enter 2nd number: 20
Invalid Input
```

Related Theory: Logic Used

What is Rotation?

A rotation is an operation similar to shift except that the bits that fall off at one end are put back to the other end. In C there is no operator for rotation. Instead rotation operation is performed by using bitwise shift operators (<< and >>) and OR operator(|).

Left shift

The expression $n << d$, the number n is moved d number of bits toward left. While shifting toward left, the left most bits will be lost and 0's will be filled at the right side.

Example

0100 1001 (decimal number 73)
Left shifting above binary number by 4 bits gives the number
1001 0000 (decimal number 144)

Right shift

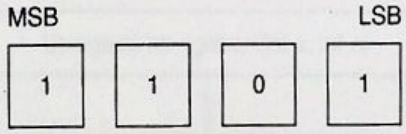
The expression $n >> d$, the number n is moved d number of bits toward right. While shifting toward right, the right most bits will be lost and 0's will be filled at the left side.

Example

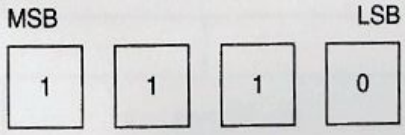
0100 1001 (decimal number 73)
Right shifting above binary number by 4 bits gives the number
0000 0100 (decimal number 4)

Right Rotation for 4-Bit Word Length

Right rotate the following no. 13 by 1 bit
13 in binary form is: 1101



After rotating right by 1 bit



Main logic for right rotation

1. Right shift number by n bits
2. Left shift number by 32-n bits
3. Bitwise or the above results

Algorithm

Start
Step 1: Read a number x
Step 2: Read the number of bits to be rotated n

Step 3: Call the rightrot function

result ← rightrot(x, n)

Step 4: Print result

Stop

Algorithm to Rotate x by n Bits

Start

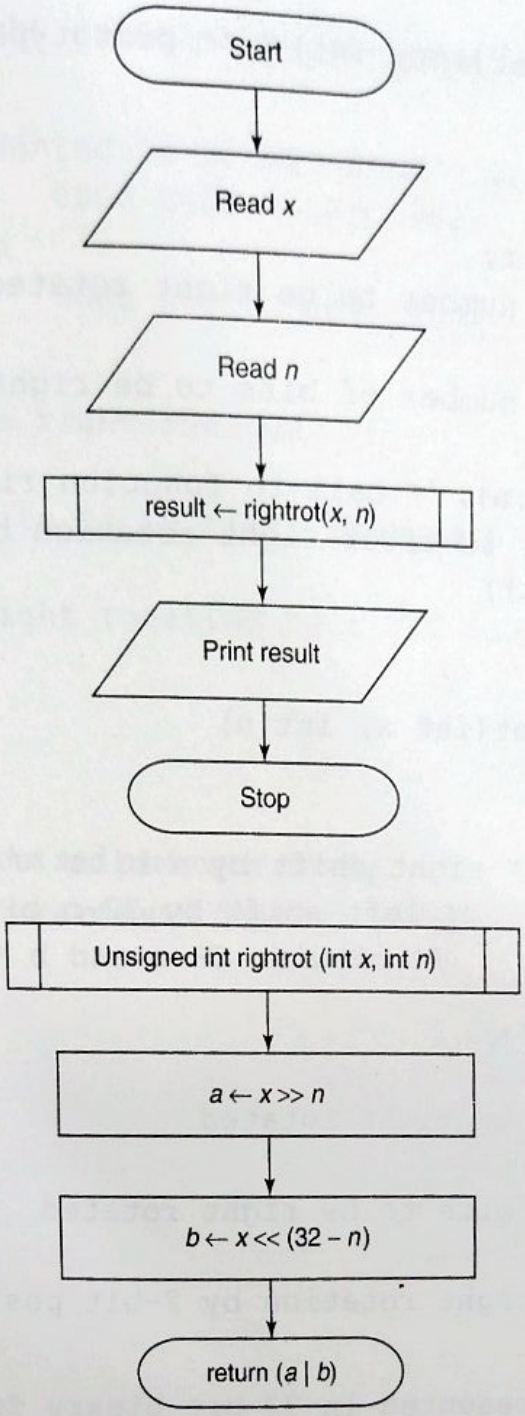
Step 1: Perform right rotation of x by n bits

$a \leftarrow x >> n$;

$b \leftarrow x << (32\text{-}n)$; /*assuming 32 bit machine*/

Step 2: return ($a \mid b$)

Stop


Start
Read x
Read n
result ← rightrot(x, n)
Print result
Stop
Unsigned int rightrot (int x, int n)
a ← x >> n
b ← x << (32 – n)
return (a | b)

Program

```
#include<stdio.h>
unsigned int rightrot(int, int); /* prototype declaration */
void main()
{
  int x, n;
  unsigned int result;
  printf("Enter the number to be right rotated \n");
  scanf("%d",&x);
  printf("Enter the number of bits to be right rotated \n");
  scanf("%d",&n);
  result=rightrot(x,n);/* call to function rightrot(x,n) */
  printf("The number %d after right rotation by %d bit positions
  is %u", x,n,result);
}

unsigned int rightrot(int x, int n)
{
  unsigned int a,b;
  a=x >> n ;        /* right shift by n bits */
  b=x << (32-n);      /* left shift by 32-n bits */
  return (a | b);       /* bitwise OR a and b */
}
```

Sample Output 1

```
Enter the number to be right rotated
8
Enter the number of bits to be right rotated
2
The number 8 after right rotation by 2-bit positions is 2
```

Conclusion

```
The number 8 is represented in 32-bit binary form as
0000 0000   0000 0000   0000 0000   0000 1000
After rotating 2-bits
0000 0000   0000 0000   0000 0000   0000 0010
```

Sample Output 2

```
Enter the number to be right rotated
9
```

```
Enter the number of bits to be right rotated
31
The number 9 after right rotation by 31-bit positions is 18
```

Conclusion

```
The number 9 is represented in 32-bit binary form as
0000 0000   0000 0000   0000 0000   0000 1001
After rotating 31-bits
0000 0000   0000 0000   0000 0000   0001 0010
```

Sample Output 3

```
Enter the number to be right rotated
16
Enter the number of bits to be right rotated
5
The number 16 after right rotation by 5-bit positions is
2147483648
```

Outcomes

- Capable of handling functions and parameter passing.
- Able to identify the importance of rotate functions and find possible applications.

Term-Work 13

Problem Definition

Design and develop a function isprime(x) that accepts an integer argument and returns 1 if the argument is prime and 0 otherwise. The function is to use plain division-checking approach to determine if a given number is prime. Invoke this function from the main with different values obtained from the user and print appropriate messages.

Objectives

- To be able to map the paper–pen method of finding prime numbers to an algorithm and then write a C code.
- To understand usage of functions.
- To understand how to optimize the logic and in turn the processing time.

Related Theory

Prime Number

A number which is only divisible by 1 and itself is called as a prime number. For example 2, 3, 5, 7, 11, etc. are not divisible by any number other than 1 and themselves, so they are all prime numbers. But the numbers such as 4, 6, 8, 16, etc. are divisible by 2 and hence they are non-prime numbers.

Program Logic

Let us see now how to check whether a number is prime or not.

- Consider number 16 which is not divisible by 9, 10, 11, 12, 13, 14 – all are greater than 16/2. But 16 is divisible by 2, 4, 6, 8 which are less than or equal to 16/2
- So the first point to remember is: A number n cannot be divided by a number that is greater than $n/2$.
- So to check if a given number (n) is prime or not, it is sufficient to divide n by 2, 3, 4, 5, …, $n/2$.

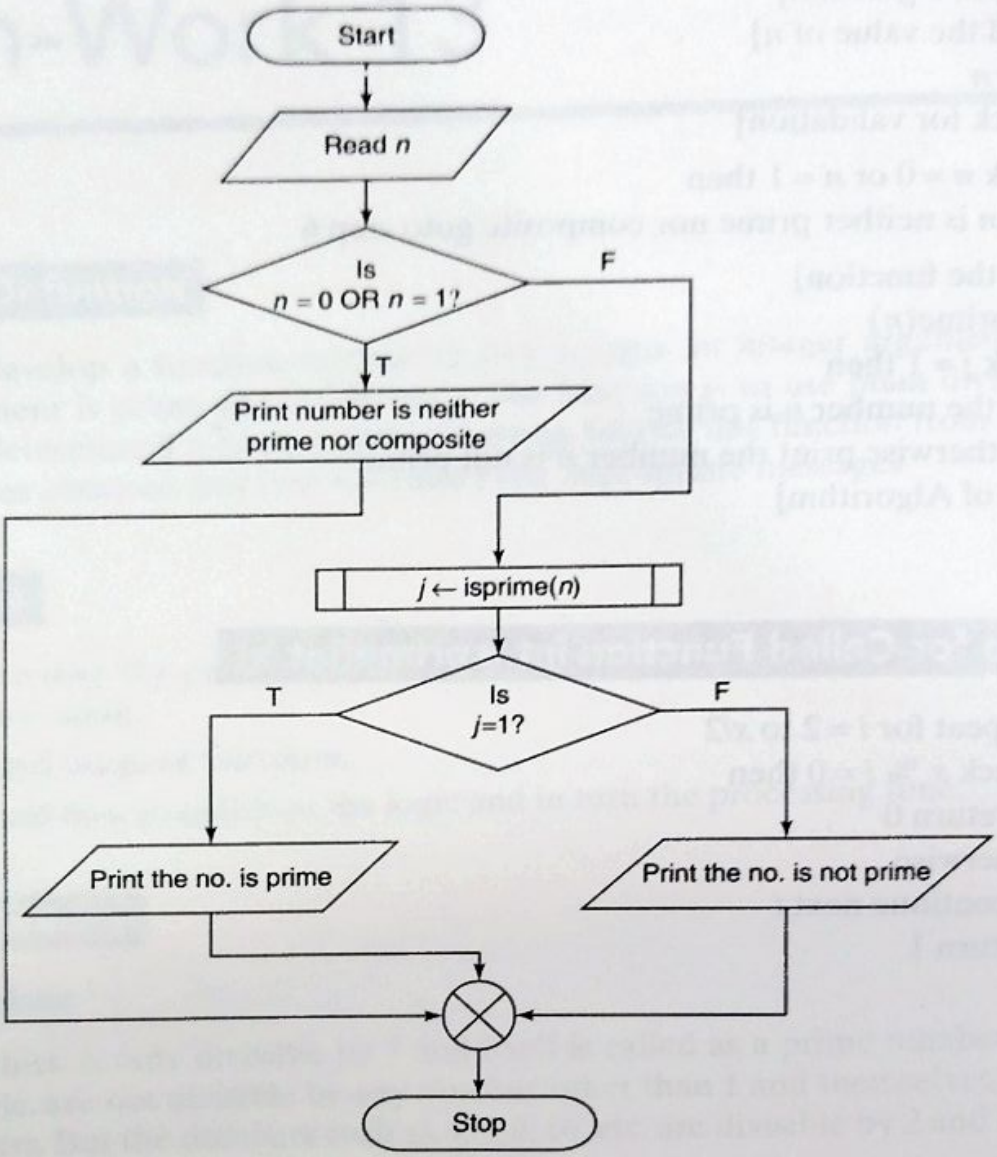
Algorithm

Step 1: [Start of algorithm]
Step 2: [Read the value of n]
Read n
Step 3: [Check for validation]
Check $n = 0$ or $n = 1$ then
print n is neither prime nor composite goto step 6
Step 4: [Call the function]
j = isprime(n)
Step 5: Check $j = 1$ then
print the number n is prime
otherwise print the number n is not prime
Step 6: [End of Algorithm]
Stop

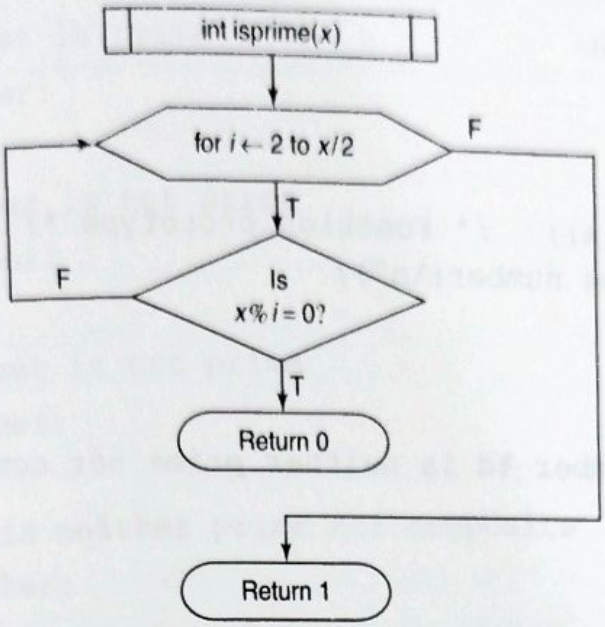
Algorithm for Called Function int isprime(int x)

Step 4.1: Repeat for $i = 2$ to $x/2$
check x % $i = 0$ then
return 0
otherwise
continue next i
Step 4.2: Return 1

Flowchart


Start
Read n
Is
n = 0 OR n = 1?
F
T
Print number is neither
prime nor composite
j ← isprime(n)
Is
j=1?
T
F
Print the no. is prime
Print the no. is not prime
Stop

/*Flowchart for Function*/



Program

```
#include<stdio.h>
#include<process.h>
void main()
{
  int j,n;
  int isprime(int x);   /* Function prototype */
  printf("Enter the number:\n");
  scanf("%d",&n);
  if(n==0||n==1)
  {
    printf("The number %d is neither prime nor composite\n",n);
    exit(0);
  }

  j = isprime(n);
  if(j==1)
    printf("The given number is prime\n");
  else
    printf("Number is not prime\n");
}

//Function to check whether the given value is prime or no

int isprime(int x)    /* Function definition */
{
  int i;     /* Local declaration  */
  for(i=2;i<=x/2;i++)
  {
    if(x%i==0)
    {
      return 0;     /* Number is not a prime */
    }
  }
  return 1;     /* Number is prime */
}
```

Sample Outputs

```
1.Enter the number:
  7
  The given number is prime
2.Enter the number:
  10
  The given number is not prime
3.Enter the number:
  16
  The given number is not prime
4.Enter the number:
  0
  The number 0 is neither prime nor composite
5.Enter the number:
  1
  The number 1 is neither prime nor composite
```

Outcomes

- Able to handle functions and passing parameters.
- Capable of optimizing the logic and code.

Term-Work 14

Problem Definition

Design, develop and execute a parallel program in C to determine and print prime numbers which are less than 100 making use of the algorithm of the Sieve of Eratosthenes.

Objectives

- To learn the concept of parallel programming and functions.
- To understand the underlying logic to find prime numbers using Sieve of Eratosthenes of algorithm and write a program for the same.

Prerequisites for Coding

OpenMP (Open Multi-Processing)

OpenMP (Open Multi-Processing) is an API (application programming interface) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor architectures and operating systems, including Linux, Unix, AIX, Solaris, Mac OS X, and Microsoft Windows platforms. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. OpenMP uses a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer.

Directives

- #pragma omp parallel for
- The parallel construct forms a team of threads and starts parallel execution.
- #pragma omp parallel for [clause[[,] clause] ...] new-line
- for-loop

An OpenMP executable directive applies to the succeeding structured block or an OpenMP Construct. A "structured block" is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.

omp_get_thread_num Function

The omp_get_thread_num function returns the thread number, within its team, of the thread executing the function. The thread number lies between 0 and omp_get_num_threads() –1, inclusive. The master thread of the team is thread 0.

The format is as follows:

```
#include <omp.h>
int omp_get_thread_num(void);
```

If called from a serial region, omp_get_thread_num returns 0. If called from within a nested parallel region that is serialized, this function returns 0.

Theory

Parallel Computing

- Traditionally, software has been written for serial computation.
- Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem.
- In this technique a large problem can be divided into smaller segments which can be solved independently and simultaneously; the computers that facilitate parallel coding are called parallel computers.

Prime Numbers

- A prime number (such as 2 or 3 or 5) is a natural number (positive whole number) which is only divisible by itself and by 1 (and no other natural number).
- The prime numbers less than 100 are:

  2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97.

Sieve of Eratosthenes Algorithm

This process is called the Sieve of Eratosthenes, after the Greek mathematician, from Alexandria, who invented it. He is also the man who first measured the circumference of the earth. This method, of listing primes, is called a "sieve" because it is like taking all of the numbers (up to some maximum) and running them through a sieve to separate out all of the primes.

Sieve of Eratosthenes Method

- There is a fairly simple method for making a list of primes. We will start with a list of all of the numbers from 2 to 100:

  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

- We now make 2 bold (you can circle it), identifying it as prime (it is not divisible by lesser primes), and cross out every second number after 2:

0 1 **2** 3 0 5 0 7 0 9 0 11 0 13 0 15 0 17 0 19 0 21 0 23 0 25 0 27 0 29 0 31 0 33 0 35 0 37 0 39 0 41 0 43 0 45 0 47 0 49 0 51 0 53 0 55 0 57 0 59 0 61 0 63 0 65 0 67 0 69 0 71 0 73 0 75 0 77 0 79 0 81 0 83 0 85 0 87 0 89 0 91 0 93 0 95 0 97 0 99 0

- We have now identified 3 as a prime. It is not divisible by lesser primes. And we cross out every third number after 3. Some of these are already crossed out. Just skip over those:

  0 1 **2 3** 0 5 0 7 0 0 0 11 0 13 0 0 0 17 0 19 0 0 0 23 0 25 0 0 0 29 0 31 0 0 0 35 0 37 0 0 0 41 0 43 0 0 0 47 0 49 0 0 0 53 0 55 0 0 0 59 0 61 0 0 0 65 0 67 0 0 0 71 0 73 0 0 0 77 0 79 0 0 0 83 0 85 0 0 0 89 0 91 0 0 0 95 0 97 0 0 0

- We do the same thing with 5, and then 7:

  0 1 2 3 0 **5** 0 **7** 0 0 0 11 0 13 0 0 0 17 0 19 0 0 0 23 0 0 0 0 0 29 0 0 0 0 0 0 37 0 0 0 41 0 43 0 0 0 47 0 0 0 0 0 53 0 0 0 0 0 59 0 0 0 0 0 0 67 0 0 0 0 73 0 0 0 0 0 79 0 0 0 83 0 0 0 0 0 89 0 0 0 0 0 0 0 97 0 0 0

- And now, we can stop! You may want to go on and try 11. But that is unnecessary. No more numbers will be crossed out, between 2 and 100
- We can stop at the square root of 100, which is 10. The reason for this is that any number less than 100 (91, for example), which is divisible by a number greater than the square root of 100 (13, in this example), is also divisible by a number less than the square root of 100 (7, in this example). So, we have already crossed out all such numbers.
- Removing the 0's, we have this list of primes:

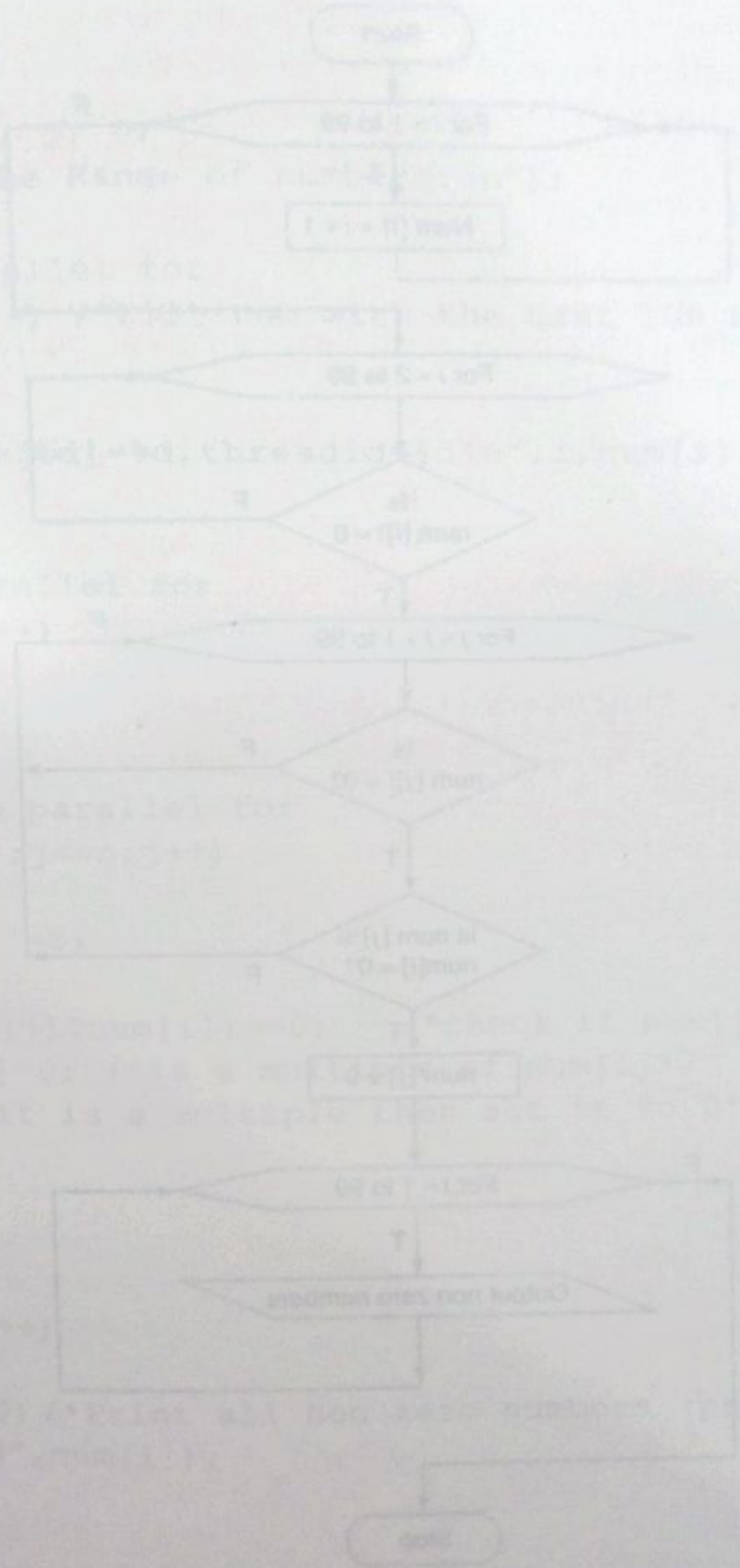
  2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

Algorithm

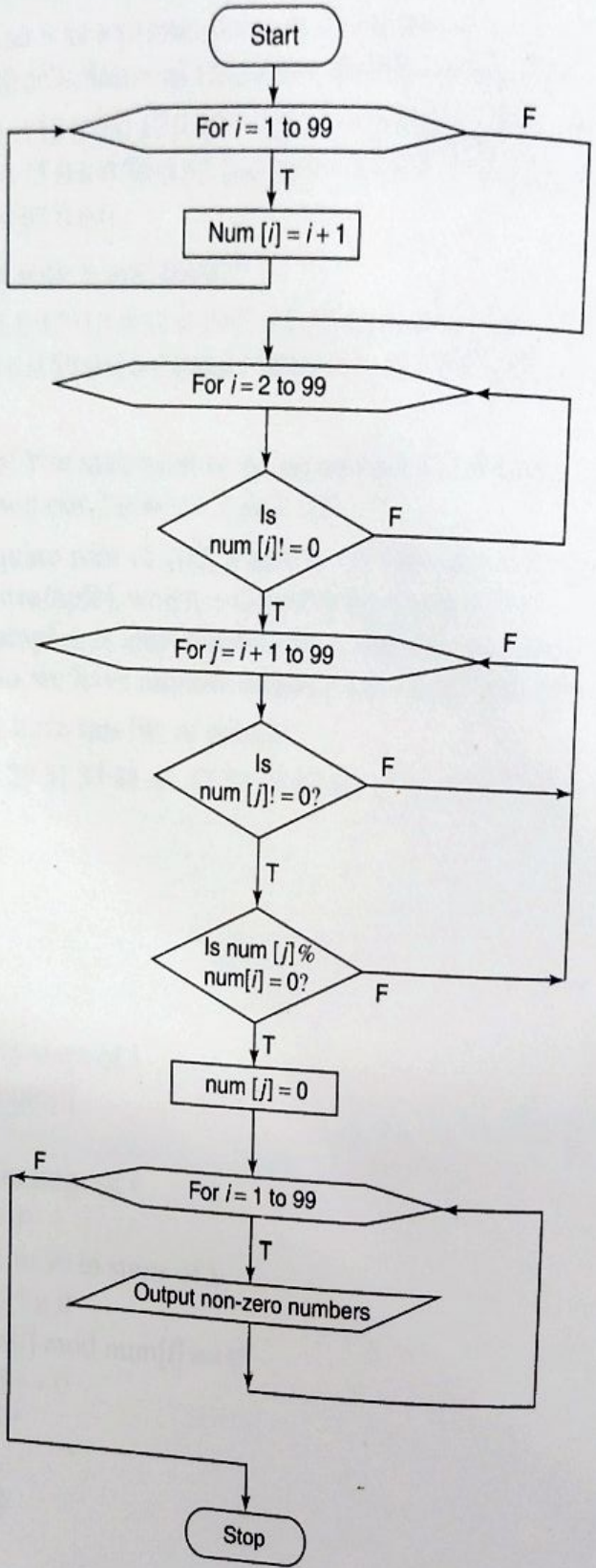
Start
Step 1: Read *n*
Step 2: For *i*<–1 to 99 in steps of 1
    Store in array num[*i*]
    END FOR
Step 3: For *i*<–2 to 99 in steps of 1
      IF num[*i*] ! = 0
        For *j*<–*i*+1 to 99 in steps of 1
          IF num[*j*] ! = 0
            IF num[*j*] mod num[*i*] == 0
              num[*j*] = 0
            END IF
          END IF
        END FOR
      END IF
    END FOR

Step 4: For i<–1 to 99 in steps of 1
Output non-zero numbers
END FOR

Stop



Flowchart


Start
For i = 1 to 99
F
T
Num [i] = i + 1
For i = 2 to 99
Is num [i]! = 0
F
T
For j = i + 1 to 99
F
Is num [j]! = 0?
F
T
Is num [j] % num[i] = 0?
F
T
num [j] = 0
F
For i = 1 to 99
T
Output non-zero numbers
Stop

Program

```
#include<stdio.h>
#include<omp.h>
void main()
{
 int num[100], i, j, n;
 printf("Enter the Range of numbers:\n");
 scanf("%d",&n);
 #pragma omp parallel for
 for(i=1;i<=n;i++) /*Fill num with the first 100 numbers*/
 {
  num[i]=i;
  printf("\nnum[%d]=%d,threadid=%d\n",i,num[i],omp_get_thread_
  num());
 }
 #pragma omp parallel for
 for(i=1;i<=n;i++)
 {
  if(num[i]!=0)
  {
   #pragma omp parallel for
   for(j=(i+1);j<=n;j++)
   {
    if(num[j]!=0)
    {
     if((num[j]%num[i])==0)    /*check if num[j]*/
      num[j]=0; /*is a multiple of num[i]*/
      /*if it is a multiple then set it to 0*/
    }
   }
  }
 }
 for(i=1;i<n;i++)
 {
  if(num[i]!=0) /*Print all non zero numbers (prime numbers)*/
   printf("%d",num[i]);
 }
}
```

Sample Output: 1

```
Enter the Range of numbers:
10
num[0]=1,threadid=0
num[6]=7,threadid=1
num[7]=8,threadid=1
num[8]=9,threadid=1
num[9]=10,threadid=1
num[10]=11,threadid=1
num[1]=2,threadid=0
num[2]=3,threadid=0
num[3]=4,threadid=0
num[4]=5,threadid=0
num[5]=6,threadid=0
  2  3  5  7
```

Sample Output: 2

```
Enter the Range of numbers:
100
num[51]=52,threadid=1
num[0]=1,threadid=0
num[52]=53,threadid=1
num[1]=2,threadid=0
num[53]=54,threadid=1
num[2]=3,threadid=0
num[54]=55,threadid=1
num[3]=4,threadid=0
num[55]=56,threadid=1
num[4]=5,threadid=0
num[56]=57,threadid=1
num[5]=6,threadid=0
num[57]=58,threadid=1
num[6]=7,threadid=0
num[58]=59,threadid=1
num[7]=8,threadid=0
num[59]=60,threadid=1
num[8]=9,threadid=0
```

```
num[60]=61,threadid=1
num[9]=10,threadid=0
num[61]=62,threadid=1
num[10]=11,threadid=0
num[62]=63,threadid=1
num[11]=12,threadid=0
num[63]=64,threadid=1
num[12]=13,threadid=0
num[64]=65,threadid=1
num[13]=14,threadid=0
num[65]=66,threadid=1
num[14]=15,threadid=0
num[66]=67,threadid=1
num[15]=16,threadid=0
num[67]=68,threadid=1
num[16]=17,threadid=0
num[68]=69,threadid=1
num[17]=18,threadid=0
num[69]=70,threadid=1
num[18]=19,threadid=0
num[70]=71,threadid=1
num[19]=20,threadid=0
num[71]=72,threadid=1
num[20]=21,threadid=0
num[72]=73,threadid=1
num[21]=22,threadid=0
num[73]=74,threadid=1
num[22]=23,threadid=0
num[74]=75,threadid=1
num[23]=24,threadid=0
num[75]=76,threadid=1
num[24]=25,threadid=0
num[76]=77,threadid=1
num[25]=26,threadid=0
num[77]=78,threadid=1
```

```
num[26]=27,threadid=0
num[78]=79,threadid=1
num[27]=28,threadid=0
num[79]=80,threadid=1
num[28]=29,threadid=0
num[80]=81,threadid=1
num[29]=30,threadid=0
num[81]=82,threadid=1
num[30]=31,threadid=0
num[82]=83,threadid=1
num[31]=32,threadid=0
num[83]=84,threadid=1
num[32]=33,threadid=0
num[84]=85,threadid=1
num[33]=34,threadid=0
num[85]=86,threadid=1
num[34]=35,threadid=0
num[86]=87,threadid=1
num[35]=36,threadid=0
num[87]=88,threadid=1
num[36]=37,threadid=0
num[88]=89,threadid=1
num[37]=38,threadid=0
num[89]=90,threadid=1
num[38]=39,threadid=0
num[90]=91,threadid=1
num[39]=40,threadid=0
num[91]=92,threadid=1
num[40]=41,threadid=0
num[92]=93,threadid=1
num[41]=42,threadid=0
num[93]=94,threadid=1
num[42]=43,threadid=0
num[94]=95,threadid=1
num[43]=44,threadid=0
```

```
num[95]=96,threadid=1
num[44]=45,threadid=0
num[96]=97,threadid=1
num[45]=46,threadid=0
num[97]=98,threadid=1
num[46]=47,threadid=0
num[98]=99,threadid=1
num[47]=48,threadid=0
num[99]=100,threadid=1
num[48]=49,threadid=0
num[100]=101,threadid=1
num[49]=50,threadid=0
num[50]=51,threadid=0
  2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59
61  67  71  73  79  83  89  97
```

Outcomes

- Capable of realizing the importance of threads in operating systems.
- Capable of identifying the programs that may be processed in parallel.

Term-Work 15

Problem Definition

Design and develop a function reverse(*s*) in C to reverse the string *s* in place. Invoke this function from the main for different strings and print the original and reversed strings.

Objectives

- To understand the fundamentals of string and character.
- To understand and use pre-defined functions.
- To manipulate characters and strings by using the pre-defined functions.
- To reverse a given string using a function reverse.

Prerequisites for Coding

String

A string is a collection of characters. Strings are always enclosed in double quotes as "string_constant."

Following are some of the string-handling operations:

- Finding the length of a string.
- Comparing two strings.
- Copying one string to another.
- Converting lower case to upper case and vice versa.
- Joining two strings.
- Reversing a string.

Declaration

The string can be declared as follows:

Syntax: char string_name[size];

Example: char name[50];

String Structure

When a compiler assigns a string to character array then it automatically appends ***null character ('\0')*** at the end of the string. Thus, size of string = original length of string.

```
char name[7];
name = "TECHNO"
```

'T'	'E'	'C'	'H'	'N'	'O'	'\0'
1	2	3	4	5	6	7

Read Strings

To read a string, we can use *scanf()* function with format specifier *%s*.

```
char name[50];
scanf("%s",name);
```

The above format allows accepting only that string which does not have any blank space, tab, new line, form feed, or carriage return.

However, C supports a library function known as *gets* that allows reading a string of text containing whitespaces. This is a simple function with one-string parameter.

```
gets(name);
```

where name is a string variable.

Write Strings

To write a string, we can use *printf()* function with format specifier *%s*.

```
char name[50];
scanf("%s",name);
printf("%s",name);
```

Another and more convenient way of printing string is to use the function *puts*. This is a simple function with one-string parameter.

```
puts(name);
```

where name is a string variable.

String Functions

"*string.h*" is a header file which includes the declarations, functions, constants of string-handling utilities. These string functions are widely used today by many programmers to deal with string operations.

Some of the standard member functions of *string.h* header files are:

Function Name	Description
strlen()	Returns the length of a string.
strlwr()	Returns upper case letter to lower case.
strupr()	Returns lower case letter to upper case.
strcat()	Concatenates two strings.
strcmp()	Compares two strings.
strrev()	Returns reverse of a string.
strcpy()	Copies a string from source to destination.

Description

Here the function reverse (str) is called by the function main( ) by passing the value of string. Function reverse(str) processes the string array by using its local variables, so as to reverse the contents. Once completed, function reverse(str) displays the result, which is reverse of the given string.

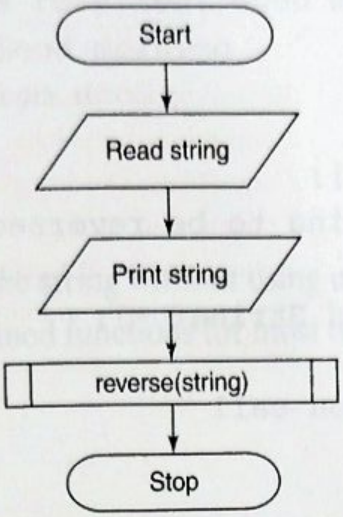
Algorithm

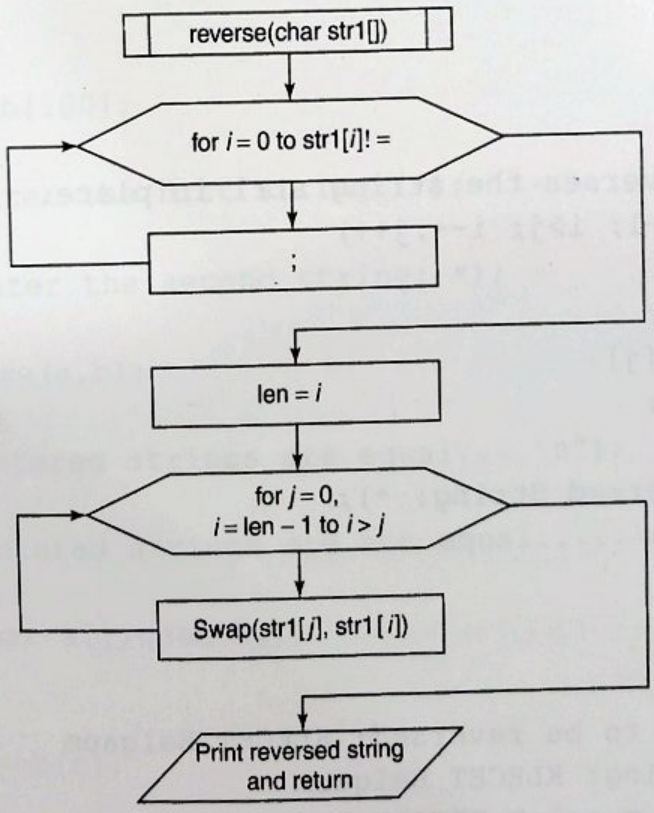
Start
Step 1: Input the string
Step 2: Print the string
Step 3: Call function reverse ()
Stop

Algorithm to reverse a string.

1. Calculate the length
   For $i = 0$ to str1[i] ≠ Null in step of 1
   end For
2. len = i
3. initialize $j = 0, i$ = len-1
4. swap(str1[j],str1[i])
5. $j = j + 1, i = i - 1$
6. repeat steps 4 to 5 until $i > j$
7. Print "reversed string"

Flowchart


Start
Read string
Print string
reverse(string)
Stop


reverse(char str1[])
for i = 0 to str1[i]! =
;
len = i
for j = 0,
i = len – 1 to i > j
Swap(str1[j], str1[i])
Print reversed string
and return

Program

```
#include<stdio.h>
void main()
{
  char str1[20];
  void reverse(char str1[]);
  printf("\n Enter the string to be reversed: ");
  gets(str1);
  printf("\n\n User Entered String: ");
  printf("%s\n",str1);
  reverse(str1); // Function call
}
void reverse(char str1[])
{
  int i,j,len,temp;
  // This loop finds the length of the string.
  for(i=0;str1[i]!='\0';i++)
  {
    ;
  }
  len=i;
  //This loop reverses the string str1 in place.
  for(j=0,i=len-1; i>j; i--,j++)
  {
    temp=str1[i];
    str1[i]=str1[j];
    str1[j]=temp;
  }
  printf("\n Reversed String: ");
  puts(str1);
}
```

Output

```
1.
Enter the string to be reversed: KLECET Belgaum
User Entered String: KLECET Belgaum
Reversed String: muagleB TECELK
2.
Enter the string to be reversed: hello
User Entered String: hello
```

```
Reversed String: olleh

3.
Enter the string to be reversed: Good morning
User Entered String: Good morning
Reversed String: gninrom dooG
```

Outcomes

- Able to optimally reverse the string without using extra variables.
- Capable of writing user-defined functions for most of the standard string library functions.

Related Programs

Program 1

C program to compare two strings without using strcmp.

```
#include<stdio.h>
#include<string.h>
void main()
{
  char a[100], b[100];
  int result;
  printf("\n Enter the first string: ");
  gets(a);
  printf("\n Enter the second string: ");
  gets(b);
  result=compare(a,b);
  if(result==0)
  printf("\n Entered strings are equal....\n");
  else
  printf("\n Entered strings are not equal.....\n");
 }
 int compare(char a[],char b[])
 {
   int c=0;
    while(a[c]==b[c])
   {
    if(a[c]=='\0'||b[c]=='\0')
    break;
    c++;
   }
```

```
  if(a[c]=='\0'&&b[c]=='\0')
  return 0;
  else
  return -1;
}
```

Output

```
1.
Enter the first string: welcome
Enter the second string: welcome
Entered strings are equal....
Press any key to continue . . .

2.
Enter the first string: computer concepts
Enter the second string: c programming
Entered strings are not equal.....
Press any key to continue . . .

3.
Enter the first string: good morning
Enter the second string: good morning
Entered strings are equal....
Press any key to continue . . .

4.
Enter the first string: good bad
Enter the second string: good
Entered strings are not equal.....
Press any key to continue . . .
```

Program 2

C program to check whether a given string is a palindrome or not.

```
#include<stdio.h>
#include<string.h>
main()
{
  char text[100];
  int begin, middle, end, length=0;
  printf("\n Enter string: ");
  gets(text);
```

```
while(text[length]!='\0')
  length++;
  end=length-1;
  middle=length/2;
  for(begin=0;begin<middle;begin++ )
  {
    if(text[begin]!=text[end])
    {
      printf("\n\n String Is Not palindrome....\n\n");
      break;
    }
    end--;
  }
  if(begin==middle)
    printf("\n\n String Is Palindrome...\n\n");
    return 0;
}
```

Output

```
1.
Enter string: madam
String Is Palindrome...
Press any key to continue . . .

2.
Enter string: programming
String Is Not palindrome....
Press any key to continue . . .
```

Program 3

C program to find the length of a string without using any built-in function.

```
#include<stdio.h>
void main()
{
  char str1[20];
  int i,count=0;
  printf("\n Enter the string: ");
  gets(str1);
  for(i=0;str1[i]!='\0';i++)
  {
    count++;
```

```
    }
        printf("\n The length of a string is:%d\n",count);
}
```

Output

```
1.
Enter the string: welcome to ccp lab
The length of a string is:18
Press any key to continue . . .

2.
Enter the string: work is worship
The length of a string is:15
Press any key to continue . . .
```

Practice Programs

1. Design and develop a C program to swap two strings.
2. Design and develop a C program to remove spaces, blanks from a string.
3. Design and develop a C program to remove vowels from a string.

Term-Work 16

Problem Definition

Design and develop a function matchany ($s1, s2$) that returns the first location in the string $s1$ where any character from the string $s2$ occurs, or –1 if $s1$ contains no character from $s2$. Do not use the standard library function that does a similar job! Invoke the function matchany ($s1, s2$) from the main for different strings and print both the strings and the return value from the function matchany ($s1, s2$).

Objectives

- Competency in using functions.
- To understand manipulation of strings.

Description

Here the function matchany($s1, s2$) is called by the function main( ) by passing the values of two character strings. Function matchany($s1, s2$) processes the string arrays using local variables, to find any character of string $s2$ matching any character in string $s1$. If any matching character is found then the function matchany($s1, s2$) will display the position of the matched character in $s1$, else –1 is displayed.

Algorithm

Start

Step 1: Input string1

Step 2: Input string2

Step 3: Call function matchany (str1, str2)

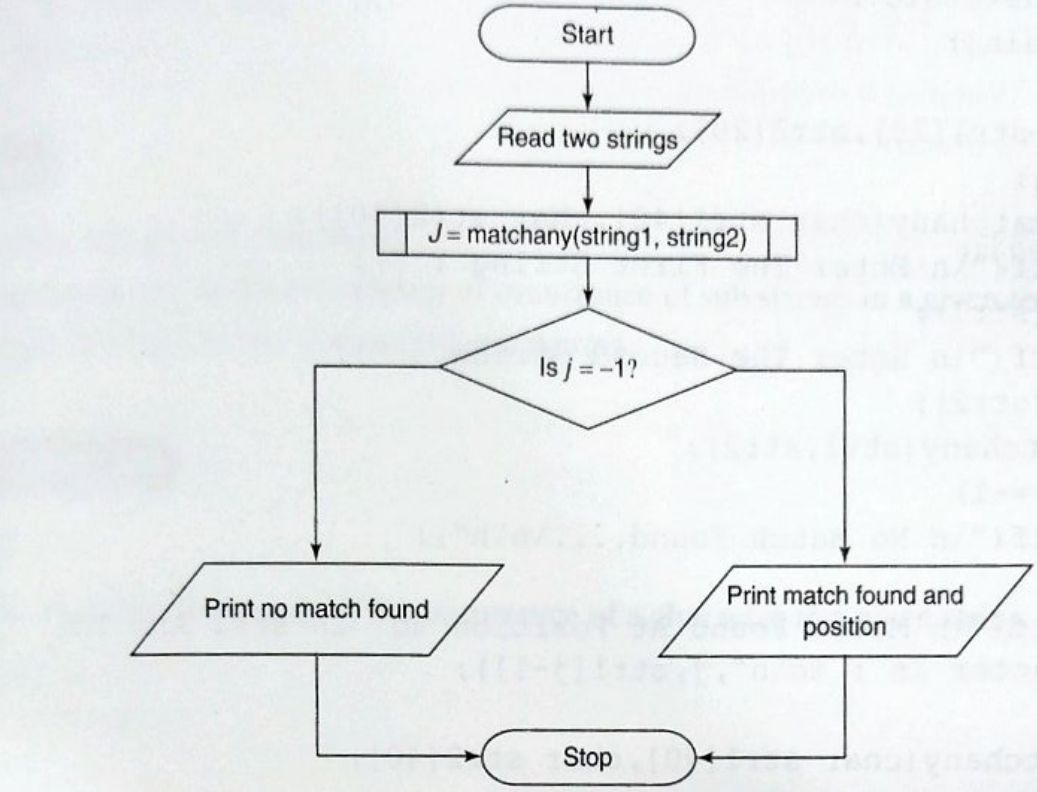
Step 4: IF $j = -1$ then
Print "No matching character found"
else
Print "Match found at position j"
END IF

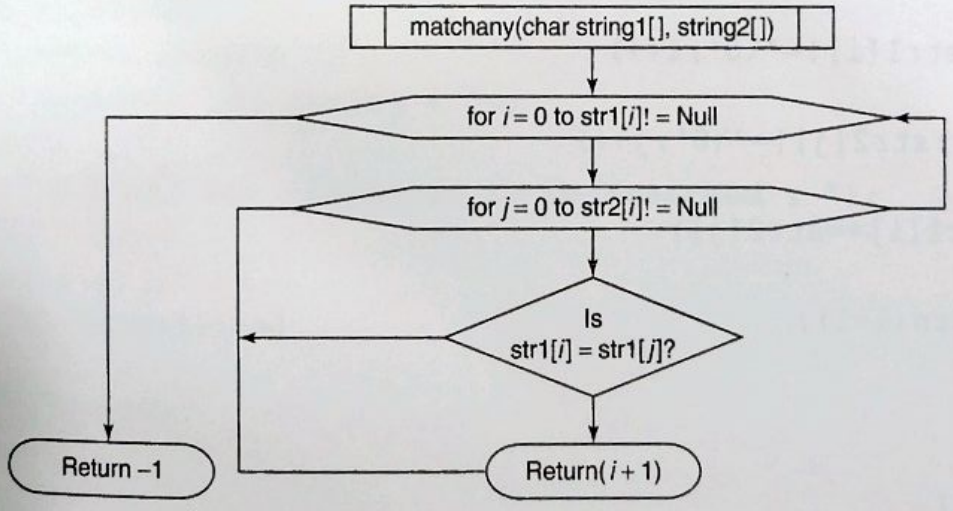
Stop

Algorithm of matchany function

1. FOR $i = 0$ to str1 $[i] \neq$ Null in steps of 1
   FOR $j = 0$ to str2 $[j] \neq$ Null in steps of 1
   IF str1 $[i]$ = str2 $[j]$ then
   Return $i + 1$
   End IF
   End For
   End For
2. Return –1

Flowchart


Start
Read two strings
J = matchany(string1, string2)
Is j = –1?
Print no match found
Print match found and position
Stop


matchany(char string1[], string2[])
for i = 0 to str1[i]! = Null
for j = 0 to str2[i]! = Null
Is str1[i] = str1[j]?
Return –1
Return(i + 1)

Program

```
#include<stdio.h>
void main()
{
  char str1[20],str2[20];
  int j;
  int matchany(char str1[40],char str2[40]);
  printf("\n Enter The First String : ");
  gets(str1);
  printf("\n Enter The Second String : ");
  gets(str2);
  j=matchany(str1,str2);
  if(j==-1)
  printf("\n No Match Found....\n\n");
  else
  printf("\n Match Found At Position %d  in str1 And The
  Character Is : %c\n",j,str1[j-1]);
}
int matchany(char str1[40],char str2[40])
{
  int i,j;
  for(i=0;str1[i]!='\0';i++)
  {
    for(j=0;str2[j]!='\0';j++)
    {
      if(str1[i]==str2[j])
      {
        return(i+1);
      }
    }
  }
  return -1;
}
```

Output

```
1.
Enter The First String : computer
Enter The Second String : game
Match Found At Position 3 in str1 And The Character Is : m
```

```
2.
Enter The First String : book
Enter The Second String : pen
No Match Found...
```

Outcomes

- Able to handle strings and functions.
- Capable of editing and finding frequency of occurrence of sub strings in a given text.
- Able to apply this logic in text formatting and editing.

Related Program

Program 1

C program to check the frequency of the occurrence of a character in a given string.

```
#include<stdio.h>
#include<string.h>
main()
{
  char str[100],ch;
  int i,count=0,length;
  printf("\n Enter The String : ");
  gets(str);
  printf("\n Enter the character to be searched : ");
  scanf("%c",&ch);
  length=strlen(str);
  for(i=0;i<length;i++)
  {
    if(str[i]==ch)
    count++;
  }
  printf("\n The frequency of the occurence of %c is %d\n\n",
  ch,count);
}
```

Output

```
1.
Enter The String : computer is an electronic device
```

```
Enter the character to be searched : e
The frequency of the occurence of e is 5
2.
Enter The String : A string is a collection of characters
Enter the character to be searched : a
The frequency of the occurence of a is 3
```

Practice Programs

1. Design and develop a C program to display the resultant string after finding a specified character.
2. Design and develop a C program to find a sub string from a given string.
3. Design and develop a C program to sort a list of names in alphabetical order.

Common C Programming Errors

1. *Missing semicolons*

   Every C statement must end with a semicolon. A missing semicolon may result in error message.

   Example: $a = x + y$

2. *Misuse of semicolons*

   Another common mistake is to put a semicolon in the wrong place.

   Example: for($i = 1$; $i <= 10$; i++);

   sum = sum + i;

3. *Undeclared variables*

   C requires every variable to be declared for its type, before it is used.

4. *Missing & operator in scanf parameters*

   All variables in a scanf call should be preceded by an **&** operator. If the variable code is declared as an integer, then the statement: *scanf ("%d",code);* is wrong.

   The correct one is: *scanf ("%d",&code);*

5. *Missing/unmatched braces*

   It is common to forget a closing brace when coding a deeply nested loop. The number of opening braces should match with the closing ones.

6. *Using a single equal sign to check equality*

   If you use a single equal sign to check equality, your program will instead assign the value on the right side of the expression to the variable on the left-hand side. Therefore, use == to check for equality.

7. *Undeclared functions*

   Every function that is called should be declared in the calling function for the types of value it returns.

8. *Missing quotes*

   Every string must be enclosed in double quotes, while a single-character constant must be enclosed in single quotes.

   Example: char ch = 'A' /* *A is a character constant* */

   char ch = "hello" /* *hello is a string* */

9. *Crossing the bounds of an array*

Arrays in C always start at index 0. This means that an array of 10 integers defined as

Example: int a[10] has valid indices from 0 to 9 not 10!

10. *Improper comment characters*

Every comment should start with a /* and end with a */. Anything between them is ignored by the compiler.

Example: .....

```
/* comment line 1
        Statement 1;
        Statement 2;
/* comment line 1 */
        Statement 3;
        Statement 4;
        .....
```

Since the closing */ is missing in the comment line1, all the statements that follow, until the closing comment */ in comment2 are ignored.

Note

1. int $x = 2$;

```
switch(x)
{
    case 2: printf("Two\n");
    case 3: printf("Three\n");
}
```

prints out:

Two

Three

Put a break after each case to come out of the switch.

2. The C compiler often does not flag an error when = is used when one really wanted an ==.

C's = operator is used exclusively for assignment and returns the value assigned. The == operator is used exclusively for comparison and returns an integer value (zero for *false*, non-zero for *true*).

Example: int $x = 5$;

```
if (x = 6)
printf("x equals 6\n");
```

This code prints out x equals 6

The assignment inside the if sets x to 6 and returns the value 6 to the if. Since 6 is not 0, this is interpreted as *true*.

3. Arrays in C always start at index 0. This means that an array of 10 integers defined as int a[10] has valid indices from 0 to 9 *not* 10!
   - It is very common among students to use a[10], to refer the last element, which may lead to segmentation fault/unpredictable error.
4. If both operands are of an integral type, integer division is used by default

   For example:

   double half = 1/2;

   This code sets half to 0 not 0.5!

   Why? Because 1 and 2 are integer constants.
   - To fix this, change at least one of them to a real constant.

     double half = 1.0/2;
   - If both operands are integer variables and real division is desired, cast one of the variables to double (or float).

     int $x = 5, y = 2$;

     double d = ((double) x)/y;
5. In C, a loop repeats the immediate next statement after the loop statement in the absence of braces.

   The code:

   int $x = 5$;

   while($x > 0$);

   x--;

   is an infinite loop. Why? The semicolon after the while defines the statement to repeat as the null statement (which does nothing). Remove the semicolon and the loop works as expected.
6. Another common loop error is to iterate one too many times or one too few. Check loop conditions carefully!

   for($i = 0; i < 10; i$++);

   Output: 10
7. Prototypes tell the compiler important features of a function: the return type and the parameters of the function.

   If no prototype is given, the compiler *assumes* that the function returns an int and can take any number having parameters of any type.
8. Include appropriate header files in the program before using the standard library functions.

   Example: double x = sqrt(2);

   include the header file math.h