



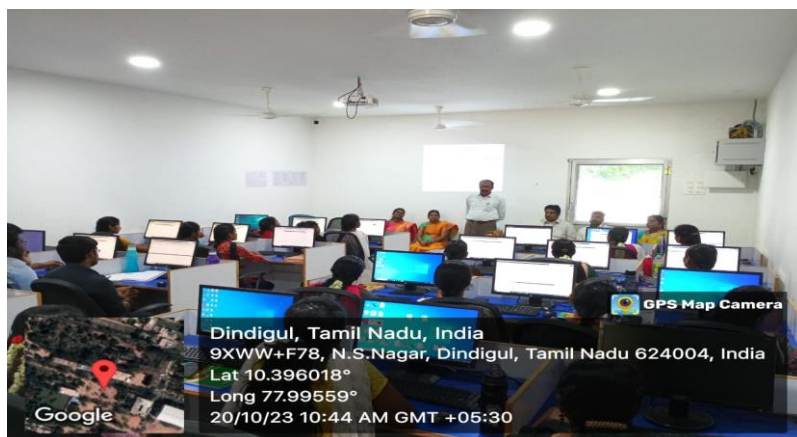
G. T. N. ARTS COLLEGE, (AUTONOMOUS)

A Telugu Linguistic Minority Institution | Affiliated to Madurai Kamaraj University

Re-accredited with 'A++' Grade by NAAC in the 2nd Cycle with CGPA 3.56

Recognized under DBT STAR COLLEGE SCHEME

Old Karur Road, Dindigul – 624 005.



DEPARTMENT OF MATHEMATICS

DBT STAR COLLEGE SCHEME

**NUMERICAL METHODS WITH C PROGRAMMING -
20UMAA5P**

LABORATORY MANUAL (2023 - 2024)

Supported by



**Department of
BioTechnology,
Government
of India**

सत्यमेव जयते

OBJECTIVES

1. To introduce students to the basic knowledge of programming fundamentals of C language.
2. To impart writing skill of C programming to the students and solving Numerical Methods problems.
3. To impart the concepts like looping, array, functions, pointers, file, structure.

INTRODUCTION ABOUT LAB

Steps involved in program development:

To develop the program in high level language and translate it into machine level language following steps have to be practiced.

1. Writing and editing the program.
2. Linking the program with the required library modules.
3. Compiling the program.
4. Executing the program.

Algorithm:-

It is a method of representing the step by step process for solving a problem. Each step is called an instruction.

Characteristics of algorithm are:

- Finiteness: It terminates with finite number of steps.
- Definiteness: Each step of algorithm is exactly defined.
- Effectiveness: All the operations used in the algorithm can be performed exactly in a fixed duration of time.
- Input: An algorithm must have an input before the execution of program begins.
- Output: An algorithm has one or more outputs after the execution of the program.

Example of algorithm to find sum of two numbers:

Step1: BEGIN

Step2: READ a, b

Step3: ADD a and band store in variable

Step4: DISPLAY c

Step5: STOP

INTRODUCTION TO C PROGRAMMING AND NUMERICAL METHODS

1. What is C Programming Language?

C is a general-purpose, structured, high-level programming language developed by Dennis Ritchie** at Bell Laboratories (AT&T), USA, in 1972.

It is one of the most widely used programming languages for scientific, mathematical, and engineering computations.

The milestones in C's development as a language a relisted below:

1. BCPL - a user friendly OS providing powerful development tools developed from BCPL c. 1967. Assembler was tedious, long and error prone.
2. A new language "B" was attempted c. 1970 by Ken Thompson at Bell Labs.
3. UNIX was developed c.1970 using "B"-DEC PDP-7 Assembly Language.
4. A totally new language "C", a successor to "B" was developed c. 1971.
5. By1973, UNIX OS was almost totally written in "C".

C provides:

- Powerful control structures (loops, conditionals)
- Efficient use of memory
- Ability to handle both numerical and logical operations
- Fast execution speed

Hence, it is often called "the mother of all modern programming languages" because languages like C++, Java, and Python evolved from C.

2. Why Use C in Numerical Methods?

Numerical Methods involve solving mathematical problems using numerical approximations rather than analytical (exact) solutions.

Examples include:

- Finding roots of equations
- Solving systems of linear equations
- Numerical differentiation and integration
- Solving differential equations

These methods require repeated computations, iterations, and approximations — which can be easily implemented using C because it:

- Is fast and efficient for arithmetic and looping operations
- Allows direct implementation of mathematical algorithms
- Provides precise control over data types and accuracy
- Can handle arrays and matrices for large data calculations
- Is portable and works across platforms (Windows, Linux, Mac)

Thus, C serves as an ideal programming language for implementing numerical algorithms.

3. Structure of a C Program

C program is a collection of several instructions where each instruction is written as a separate statement. The C program starts with a main function followed by the opening braces which indicates the start of the function. Then follows the variable and constant declarations which are followed by the statements that include input and output statements.

C program may contain one or more sections as shown below:

DOCUMENTATION SECTION

LINK SECTION

DEFINITION SECTION

GLOBAL DECLARATION SECTION

Main () Function section

```

{
    Declaration part
    Executable part
}

```

SUBPROGRAM SECTION

User defined functions

Every C program follows a standard structure:

```

#include <stdio.h>    // Header file for input/output
#include <math.h>     // Header file for mathematical functions

int main() {
    // Variable declarations
    // Input statements
    // Processing (algorithm implementation)
    // Output statements
    return 0;
}

```

Explanation:

#include <stdio.h>- Standard Input/Output library (for printf, scanf)

#include <math.h>- Library containing math functions like sqrt(), pow(), fabs()

main()- Entry point of every C program

{ ... } - Block of statements

return 0; - Indicates successful program termination

4. Basic Steps in C Programming

Step	Action	Description
1	Write the code	Type the program in any text editor or IDE
2	Save the file	Save as program_name.c

3	Compile	Convert the code into machine language using a compiler
4	Debug	Fix any syntax or logical errors
5	Execute	Run the compiled program to see the output

5. How to Install C Compiler and IDE

A. On Windows (using Code::Blocks IDE)

1. Download Code::Blocks with MinGW from
<https://www.codeblocks.org/downloads>
2. Install it using default settings.
3. Open Code::Blocks → *File → New → Project → Console Application → C*.
4. Write your C code, then click Build and Run (F9).

B. On Windows (using Turbo C++)

1. Download Turbo C++ from trusted educational sites (e.g.,
<https://turboc.codeplex.com/> or
<https://www.turboc7.com/>).
2. Install and run Turbo C++.
3. Type the code → Save → Press Alt + F9 to compile → Ctrl + F9 to run → Alt + F5 to view output.

6. Input and Output Functions in C

Function	Purpose	Example
printf()	To display output	printf("Sum = %d", s);
scanf()	To take input	scanf("%d", &n);

Common Format Specifiers:

%d → integer

%f → float

%c → character

%s → string

7. Mathematical Functions in C

Function	Description	Example
sqrt(x)	Square root	sqrt(9) = 3
pow(x, y)	(x ^y)	pow(2,3) = 8
fabs(x)	Absolute value	fabs(-5) = 5
sin(x), cos(x), tan(x)	Trigonometric functions	sin(3.14/2) = 1
exp(x)	(e ^x)	exp(1) = 2.718
log(x)	Natural log	log(10) = 2.3026

Note: Include `<math.h>` to use these functions.

8. Errors in Numerical Computation

Numerical methods often give approximate results. Common types of errors include:

1. Round-off Error: Due to limited precision of numbers in the computer.
2. Truncation Error: Caused by approximating an infinite process (like a series).
3. Absolute Error: $(|\text{True value} - \text{Approximate value}|)$
4. Relative Error: $(\frac{|\text{True} - \text{Approximate}|}{|\text{True}|})$

Proper understanding of errors helps in writing accurate and stable C programs for numerical computation.

9. Applications of Numerical Methods in Mathematics

- Solving algebraic and transcendental equations
- Matrix computations in linear algebra
- Polynomial interpolation and regression
- Calculus (numerical differentiation/integration)
- Differential equations modeling population, heat, and motion

- Curve fitting and data analysis

10. Laboratory Setup

Software Required:

- Turbo C / Code::Blocks / Dev C++ / GCC

Hardware Requirements:

- Desktop or Laptop with minimum 2 GB RAM
- Windows / Linux / macOS Operating System

Output Format:

- **Clear program listing**
- Input and output displayed
- Result written as “Verified successfully”

11. Basic Syntax of C Programming

C is a structured programming language, meaning every program is organized into blocks and statements that follow a definite structure.

The syntax defines how instructions must be written so that the compiler can understand and execute them correctly.

12. Structure of a C Program

Every C program follows this standard format:

```
#include <stdio.h>    // Header file for input and output functions
#include <math.h>      // Header file for mathematical functions

int main() {
    // Variable declarations
    // Input statements
    // Processing statements
    // Output statements
    return 0;
}
```

Explanation

#include <stdio.h> → Allows use of printf() and scanf().

#include <math.h> → Includes math functions like sqrt(), pow(), etc.

main() → Every C program must have one main() function.

{ ... } → Denotes a block of statements.

return 0; → Indicates successful completion of the program.

12. Tokens in C

C programs are made up of tokens – the smallest building blocks.

Token Type	Example	Description
Keywords	int, float, if, while, return	Reserved words with fixed meanings
Identifiers	x, total, sum1	Names for variables or functions
Constants	5, 3.14, A	Fixed values that don't change
Operators	+, -, *, /, %, ==	Used for calculations and comparisons
Punctuators	{ } [] () , ;	Structural symbols used in code

13. Variables and Data Types

Declaration:

int a; // integer

float b; // decimal number

char c; // character

Initialization:

int a = 10;

float b = 5.6;

char c = 'A';

Common Data Types:

Data Type	Size (bytes)	Example	Format Specifier
int	2 or 4	25	%d
float	4	3.14	%f
double	8	3.14159	%1f
char	1	A	%c

14. Input and Output in C

```
#include <stdio.h>

int main() {
    int a;

    printf("Enter a number: ");
    scanf("%d", &a);
    printf("You entered: %d", a);
    return 0;
}
```

Output Example:

Enter a number: 10

You entered: 10

15. Operators in C

Arithmetic Operators

+ - * / %

Relational Operators

== != > < >= <=

Logical Operators

&& || !

Assignment Operators

`= += -= *= /=`

Increment/Decrement Operators

`++ --`

16. Control Statements

Decision Making

`if(condition)`

`statement;`

`else`

`statement;`

Looping

`for(i = 1; i <= n; i++)`

`printf("%d", i);`

While Loop

`while(condition) {`

`// statements`

`}`

Do-While Loop

`do {`

`// statements`

`} while(condition);`

17. Comments in C

Used to make notes in programs (ignored by compiler).

`// This is a single-line comment`

`/* This is a`

`multi-line comment */`

18. Example C Program

`#include <stdio.h>`

```

int main() {
    int a, b, sum;

    printf("Enter two numbers: ");
    scanf("%d%d", &a, &b);

    sum = a + b;

    printf("Sum = %d", sum);
    return 0;
}

```

Output

Enter two numbers: 5 10

Sum = 15

19. Common Syntax Errors

Type	Example	Correction
Missing semicolon	printf("Hello")	printf("Hello");
Missing brackets	if(x>0) printf("Pos");	if(x>0) { printf("Pos"); }
format specifier	scanf("%f", &x); for integer	scanf("%d", &x);

20. Salient Features of C:

C language has some characteristics that define the language and also have led to its popularity as a programming language.

- Small size.
- Extensive use of function calls.
- Structured language.
- Low level (Bitwise) programming readily available.
- Pointer implementation-extensive use of pointers for memory, array, structures and functions.

- It has high-level constructs.
- It can handle low-level activities.
- It produces efficient programs.
- It can be compiled on a variety of computers.

21. COURSE OUTCOME

After completing this lab course Students will be able to:

1. Understand the logic for a given problem.
2. Write the algorithm of a given problem.
3. Draw a flowchart of a given problem.
4. Recognize and understand the syntax and construction of C programming code.
5. Gain experience of procedural language programming.
6. Know the steps involved in compiling, linking and debugging C code.
7. Understand using header files.
8. Learn the methods of iteration or looping and branching.
9. Make use of different data-structures like arrays, pointers, structures and files.
10. Understand how to access and use library functions.
11. Understand function declaration and definition.
12. Understand proper use of user defined functions.
13. Write programs to print output on the screen as well as in the files.
14. Apply all the concepts that have been covered in the theory course, and
15. Know the alternative ways of providing solution to a given problem.

INDEX

PROG. NO.	NAME OF PROGRAM	PAGE NO.
1	Write a Program to find a root of a nonlinear equation using the Method of Bisection	1
2	Write a Program to find a root of a nonlinear equation using the Method of False Position.	5
3	Write a Program to find the root of a nonlinear equation using the Newton-Raphson method.	8
4	Write a Program to obtain the solution of a system of linear equations using Gauss elimination method.	11
5	Write a Program to obtain the solution of a system of linear equations using Gauss -Seidel method.	13
6	Write a Program to construct Newton's forward difference interpolation polynomial.	16
7	Write a Program to construct Lagrange's interpolation polynomial formula.	18
8	Write a Program to evaluate a definite integral by Trapezoidal rule.	20
9	Write a Program to evaluate a definite integral by Simpson's rule.	22
10	Write a Program to find solution of initial value problem using fourth order Runge Kutta method	24

PROGRAM 1: METHOD OF ITERATION

AIM:

Solution of Non-linear equation in single variable using the method of successive bisection.

ALGORITHM:

1. Decide initial values of a & b and stopping criterion E.
2. Compute $f_1=f(a)$ & $f_2=f(b)$.
3. If $f_1*f_2>0$, a & B do not have any root and go to step 7; otherwise continue
4. Compute $*x=(x_1+x_2)/2$ and $f_0=f(*f)$
5. If $f_1 * f_0 < 0$ then
 Set $b=*x$
 Else
 Set $a=*x$
 Set $f_1=f_2$
6. If absolute value of $b-a/b$ is less than error E, then
 Root $=(x_1+x_2)/2$
 Write the value of root
 Go to step 7
 Else
 Go to step 4
7. Stop

PROGRAM:

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

```

float x1, x2, x3;
float f1, f2, f3;
int i=0;
printf ("Enter the value of x1 & x2");
scanf ("%f%f", &x1, &x2);
do
{
f1=x1*x1*x1-3*x1+1;
f2=x2*x2*x2-3*x2+1;
x3=(x1+x2)/2;
f3=x3*x3*x3-3*x3+1;
if (f1*f3<0)
{
x2=x3;
}
else
{
x1=x3;
}
i++;
printf ("\n Iteration number=%d", i);
printf ("\n Root of function=%f", x3);
printf ("\n Value of the function=%f", f3);
}
while (fabs(f3)>0.0001);
getch ();
}

```

OUTPUT:

Enter the value of x_1 & x_2

2

Iteration number=1

Root of function=1.500000

Value of the function=-0.125000

Iteration number=2

Root of function=1.750000

Value of the function=1.109375

Iteration number=3

Root of function=1.625000

Value of the function=0.416016

Iteration number=4

Root of function=1.562500

Value of the function=0.127197

Iteration number=5

Root of function=1.531250

Value of the function=-0.003387

Iteration number=6

Root of function=1.546875

Value of the function=0.060772

Iteration number=7

Root of function=1.539063

Value of the function=0.028410

Iteration number=8

Root of function=1.535156

Value of the function=0.012441

Iteration number=9

Root of function=1.533203

Value of the function=0.004509

Iteration number=10

Root of function=1.532227

Value of the function=0.000556

Iteration number=11

Root of function=1.531738

Value of the function=-0.001416

Iteration number=12

Root of function=1.531982

Value of the function=-0.000430

Iteration number=13

Root of function=1.532104

Value of the function=0.000063

PROGRAM 2: METHOD OF FALSE POSITION

AIM:

Solution of Non-linear equation in single variable using the method of false position.

ALGORITHM:

```
# include<stdio.h>
#include<conio.h>
#include<math.h>
# define e 0.0001
# define f(x) x*x*x-4*x+1
void main ()
{
int i=0;
float x0, x1, x2, f0, f1, f2;
printf ("Enter the value of x0 and x1");
scanf ("%f%f",&x0,&x1);
do
{
f0=f(x0);
f1=f(x1);
f2=f(x2);
x2=((x0*f1)-(x1*f0))/(f1-f0);
if (f0*f2<0)
{
x1=x2;
f1=f2;
```

```

}
else
{
x0=x2;
f1=f2;
}
i++;
printf ("\nNumber of Iteration=%d", i);
printf ("\nRoot of the function=%f", x2);
printf ("\nValue of function=%f", f2);
}
while (fabs(f2)>e);
getch();
}

```

OUTPUT:

Enter the value of x0 and x1

2

Number of Iteration=1

Root of the function=1.666667

Value of function=1.000000

Number of Iteration=2

Root of the function=2.384616

Value of function=-1.037037

Number of Iteration=3

Root of the function=1.789560

Value of function=5.021399

Number of Iteration=4

Root of the function=1.875625

Value of function=-0.427132

Number of Iteration=5

Root of the function=1.857939

Value of function=0.095888

Number of Iteration=6

Root of the function=1.861369

Value of function=-0.018268

Number of Iteration=7

Root of the function=1.860696

Value of function=0.003597

Number of Iteration=8

Root of the function=1.860827

Value of function=-0.000704

Number of Iteration=9

Root of the function=1.860802

Value of function=0.000138

Number of Iteration=10

Root of the function=1.860807

Value of function=-0.000026

PROGRAM 3: GAUSS ELIMINATION METHOD

AIM:

Solution of system of linear equations using Gauss Elimination method.

ALGORITHM:

```
# include<stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    int n;
    float a[10][10], x[10], t, s;
    int i, j, k;
    clrscr();
    printf ("\n Enter number of unknowns:");
    scanf ("%d", &n);
    printf (" \n Enter the elements of augmented matrix row wise \n");
    for (i=0; i<n; i++)
        for (j=0; j<n+1; j++)
            scanf ("%f", &a[i][j]);
    for (j=0; j<n-1; j++)
        for (i=j+1; i<n; i++)
        {
            t=a[i][j]/a[j][j];
            for (k=0; k<n+1; k++)
                a[i][k] -=a[j][k]*t;
        }
```

```

printf ("\n The upper triangular matrix\n");
for (i=0; i<n; i++)
{
printf ("\n");
for (j=0; j<n+1; j++)
printf ("%8.4f", &a[i][j]);
}
for (i=n-1; i>0; i--)
{
s=0;
for(j=i+1; j<n; j++)
s +=a[i][j]*x[j];
x[i]=(a[i][n]-s)/a[i][i];
}
printf ("\n\n solution is:");
for (i=0; i<n; i++)
printf ("\n x[%d]=%7.4f",i+1,x[i]);
getch();
return (1);
}

```

OUTPUT:

Enter number of unknowns:3

Enter the elements of augmented matrix row wise

1 1 1 9

2 -3 4 13

3 4 5 40

The upper triangular matrix

0.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000

solution is:

x[1]= 0.0000

x[2]= 3.0000

x[3]= 5.0000

PROGRAM 4:NEWTON RAPHSOON METHOD

AIM:

Solution of a system of non-linear equation using Newton's Raphson Method

ALGORITHM:

```
# include <stdio.h>
# include<conio.h>
# include<math.h>
# define e 0.0001
# define f(x) x*x*x-4*x+1
# define df(x) 3*x*x-4
int main()
{
float x0,x1,f0,f1,df0;
int i=0;
printf ("Enter the value of x0");
scanf ("%f",&x0);
do
{
f0=f(x0);
df0=df(x0);
x1=x0-(f0/df0);
f1=f(x1);
x0=x1;
i++;
printf ("\nNumber of Iterations=%d", i);
printf ("\nRoot of the function=%f",x1);
```

```
printf ("\nValue of the function=%f",f1);  
}  
while ( fabs(f1)>e);  
return (0);  
}
```

OUTPUT:

Enter the value of x00

Number of Iterations=1

Root of the function=0.250000

Value of the function=0.015625

Number of Iterations=2

Root of the function=0.254098

Value of the function=0.000013

PROGRAM 5: GAUSS SEIDAL METHOD

AIM:

Solution of a system of Linear equations using Gauss Seidel method.

ALGORITHM:

```
# include <stdio.h>
# include<conio.h>
# include<math.h>

/* Arrange system of linear equations to be solved in diagonally dominant form and
form equation for each unknown and define here*/

/* In this example we are saying
 $3x+20y-z = -18$ 
 $2x-3y+20z = 25$ 
 $20x+y-2z = 17$  */

/* Arranging given systems of linear equations in diagonally dominant form:
 $20x+y-2z = 17$ 
 $3x+20y-z = -18$ 
 $2x-3y+20z = 25$  */

/* Equations:
 $X=(17-y+2z)/20$ 
 $Y=(-18-3x+z)/20$ 
 $Z=(25-2x+3y)/20$  */

/* Defining function*/
#define f1(x,y,z)(17-y+2*z)/20
#define f2(x,y,z)(-18-3*x+z)/20
#define f3(x,y,z)(25-2*x+3*y)/20

/* Main function */
int main()
{
```

```

float x0=0,y0=0,z0=0,x1,y1,z1,e1,e2,e3,e;
int count=1;
printf ("enter tolerable error:\n");
scanf ("%f", &e);
printf ("\n count \tx\ty\tz\n");
do
{
/* Calculation*/
x1=f1(x0, y0, z0);
y1=f2(x1, y0, z0);
z1=f3(x1, y1, z0);
printf("%d\t%0.4f\t%0.4f\t%0.4f\n", count, x1, y1, z1);
/* Error*/
e1=fabs(x0-x1);
e2=fabs(y0-y1);
e3=fabs(x0-x1);
count++;
/* Set value for next iteration*/
x0=x1;
y0=y1;
z0=z1;
}
while(e1>e && e2>e && e3>e);
printf("\n Solution:x=%0.3f, y=%0.3f and z=%0.3f",x1, y1,z1);
getch();
return 0;
}

```

OUTPUT:

Enter tolerable error:

0.0001

count x y z

1 0.8500 -1.0275 1.0109

2 1.0025 -0.9998 0.9998

3 1.0000 -1.0000 1.0000

4 1.0000 -1.0000 1.0000

Solution: $x=1.000$, $y=-1.000$ and $z=1.000$

PROGRAM 6:NEWTON'S FORWARD INTERPOLATION FORMULA

AIM:

To construct Newton's Forward difference Interpolation formula.

ALGORITHM:

```
# include <stdio.h>
# include<conio.h>

void main ()
{
float ax[30], ay[30], diff[30][10], nr=1.0, dr=1.0, x, p, h, yp;
int n, j, i, k;
clrscr();
printf(" \n Enter the value of n :");
scanf ("%d", &n);
printf (" \n \n Enter the value of x and y:\n");
for (i=0; i<n; i++)
scanf ("%f%f", &ax[i], &ay[i]);
printf (" \n \n Enter the value of x for which y is wanted:");
scanf ("%f", &x);
h=ax[1]-ax[0];
for (i=0; i<=n; i++)
diff[i][j]=ay[i+1]-ay[i];
for (j=2; j<n; j++)
for (i=0; i<=n-j; i++)
diff [i][j]=diff[i+1][j-1]-diff[i][j-1];
i=0;
while (!(ax[i]>x)) i++;
i--;
```

```

p = (x-ax[i]>x)/h;
yp = ay[i];
for (k=1; k<n; k++)
{
nr *=p-k+1;
dr *=k;
yp +=(nr/dr)*diff[i][k];
}
printf ("\n when x=%f, y=%f\n", x, yp, p);
getch ();
}

```

OUTPUT:

Enter the value of n :4

Enter the value of x and y:

75 246

80 202

85 118

90 40

Enter the value of x for which y is wanted:79

when x=79.000000, y=246.000000

PROGRAM 7:LAGRANGE'S INTERPOLATION FORMULA

AIM:

Construct a Lagrange's Interpolation Polynomial formula.

ALGORITHM:

```
#include<stdio.h>
#include<conio.h>
#define Max 100
void main()
{
float ax[max],ay[max], nr, dr, x, y=0;
int i, j, n;
clrscr ();
printf ("\n Enter the value of n:");
scanf ("%d", &n);
printf ("\n Enter the set of values of x and y\n");
for (i=0; i<n; i++)
scanf ("%f%f", &ax[i], &ay[i]);
printf ("\n Enter the value of x for which value of y is wanted:");
scanf ("%f", &x);
for (i=0; i<n; i++)
{
nr = dr=1;
for (j=0; j<n; j++)
if (j!=i)
{
nr *=x-ax[j];
dr *=ax[i]-ax[j];
}
```

```
}  
y +=(nr/dr)*ay[i];  
}  
printf ("\n when x=%f, y=%f \n", x, y);  
getch (); }
```

OUTPUT:

Enter the value of n:4

Enter the set of values of x and y

3 168

7 120

9 72

10 63

Enter the value of x for which value of y is wanted:6

when x=6.000000, y=147.000000

PROGRAM 8:TRAPEZOIDAL RULE

AIM:

To evaluate a definite integral by Trapezoidal rule.

ALGORITHM:

```
# include <stdio.h>
# include<conio.h>
# include<math.h>

float y(float x)
{
    return (1/(1+x*x));
}

int main()
{
    float x0, xn, h, s;
    int i, n;
    clrscr ();
    printf ("\n Enter x0:");
    scanf ("%f", &x0);
    printf ("\n Enter xn:");
    scanf ("%f", &xn);
    printf ("\n Enter No. of intervals:");
    scanf ("%d", &n);
    h = (xn-x0)/n;
    s = y(x0)+y(xn);
    for (i=1; i<n; i++);
    s +=2*y(x0+i*h);
    printf ("\n Value of the integral is %6.4f", (h/2)*s);
```

```
getch ();  
return (1);  
}
```

OUTPUT:

Enter x0:0

Enter xn:1

Enter No. of intervals:5

Value of the integral is 0.2500

PROGRAM 9:SIMPSON'S ONE THIRD RULE

AIM:

To evaluate a definite integral by Simpson's one third rule.

ALGORITHM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

float y(float x)
{
    return (1/(1+x*x));
}

int main ()
{
    float x0, xn, h, s;
    int i, n;

    clrscr ();
    printf ("\n Enter x0:");
    scanf ("%f",&x0);
    printf ("\n Enter xn:");
    scanf ("%f", &xn);
    printf ("\n Enter number of Iterations of intervals ( an even integral:");
    scanf ("%d", &n);
    h = (xn-x0)/n;
    s=y(x0) + y(xn)+4*y(x0+h);
    for (i=3; i<n; i+=2)
        s += 4*y(x0+i*h) + 2*y(x0+(i-1)*h);
    printf ("\n value of the integral is %6.4f",(h/3)*s);
```

```
getch ();  
return (1);  
}
```

OUTPUT:

Enter x0:0

Enter xn:1

Enter number of Iterations of intervals (an even integral):4

value of the integral is 0.7854

PROGRAM 10:RUNGE-KUTTA METHOD OF FOURTH ORDER

AIM:

Solution of intial value problem using foruth order Runge-Kutta method.

ALGORITHM:

```
#include<stdio.h>
#include<conio.h>
float f(float x, float y)
{
return (x*y+y*y);
}
int main()
{
float x0, y0, h, xn, x, y, k1, k2, k3, k4, k,i=0;
printf ("\n Enter the value of x0:");
scanf ("%f", &x0);
printf ("\n Enter the value of y0:");
scanf( "%f", &y0);
printf ("\n Enter the value of h:");
scanf ("%f", &h);
printf ("\n Enter the value of xn:");
scanf ("%f", &xn);
x=x0; y=y0;
while (1)
{
if ( x==xn) break;
k1=h*f(x,y);
```

```

k2=h*f(x+h/2.0, y+k1/2.0);
k3=h*f(x+h/2.0, y+k2/2.0);
k4=h*f(x+h, y+k3);
k=(k1+2*(k2+k3)+k4)/6.0;
x +=h; y +=k;
i++;
printf ("\n when x=%8.4f y=%8.4f",x,y);
}
getch ();
return 0;
}

```

OUTPUT:

Enter the value of x0:0

Enter the value of y0:1

Enter the value of h:0.1

Enter the value of xn:0.4

when x= 0.1000 y= 1.1169

when x= 0.2000 y= 1.2774

when x= 0.3000 y= 1.5041

when x= 0.4000 y= 1.8389