********************************************************************************

NAME : Pradip S Karmakar

ROLL NO : 10

CLASS : MCA-II

SUBJECT : CONM

********************************************************************************

Q(1): Evaluate Integral of (e^x^2)*sin x dx from 0 to 1 using Trapezoidal rule correct to 3 decimal places

********************************************************************************

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define epsilon 0.0005

void trapezoidal(double,double,int);

double f(double x)
{
	return (exp(x*x)*sin(x));
}

void main()
{
	int N=2;
	double a,b;
	a=0;
	b=1;
	trapezoidal(a,b,N);
	getch();
}
```

```c
void trapezoidal(double a,double b,int N)
{
	int i,limit=20,k=1;
	double sum=0,old_sum=0,h;
	printf("===================================Trapezoidal Rule=================================\n\n");
	printf("\nSr No\t\t|\tN\t\t|\th\t\t\t|\tIntegral\n");
	printf("__________________________________________________________________________________________________
______________________________");
	while(k<=limit)
	{
		sum=0;
		h=(b-a)/N;
		for(i=1;i<N;i++)
		{
			sum+=2*f(a+i*h);
		}
		sum+=(f(a)+f(b));
		sum *=h/2;
		printf("\n%d\t\t|\t%d\t\t|\t%lf\t\t|\t%lf",k,N,h,sum);
		if(fabs(sum-old_sum)<epsilon)
		{
			printf("\n\n-->The Estimate of the Integral is %lf",sum);
			break;
		}
		N*=2;
		k++;
		old_sum=sum;
	}
}
```

********************************************************************************

output:

===================================Trapezoidal Rule==================================

| Sr No | N | h | Integral |
|---|---|---|---|
| 1 | 2 | 0.500000 | 0.879636 |
| 2 | 4 | 0.250000 | 0.804736 |
| 3 | 8 | 0.125000 | 0.785295 |
| 4 | 16 | 0.062500 | 0.780386 |
| 5 | 32 | 0.031250 | 0.779156 |
| 6 | 64 | 0.015625 | 0.778848 |

-->The Estimate of the Integral is 0.778848

********************************************************************************

*********************************************************************************

Q(2): Evaluate the integral:

integral of dx/(1+x) from 0 to 1

Using

(i) Simpson's 1/3 Rule correct to six decimal places

(ii) Simpson's 3/8 rule correct to six decimal places

*********************************************************************************

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

#define epsilon 0.0000005


void simpsons1_3(double,double,int);

void simpsons3_8(double,double,int);

double f(double x)

{

	return (1/(1+x));

}


void main()

{

	int N=2;

	double a,b;

	a=0;

	b=2;

	simpsons1_3(a,b,N);

	simpsons3_8(a,b,N);

	getch();

}
```

```
void simpsons1_3(double a,double b,int N)
{
	printf("====================================Simpsons 1/3 Rule==================================\n\n");
	int i,limit=20,k=1;
	double sum=0,old_sum=0,h;
	printf("\nSr No\t\t|\tN\t\t|\th\t\t\t|\tIntegral\n");
	printf("____________________________________________________________________________________________________");
	while(k<=limit)
	{
		sum=0;
		h=(b-a)/N;
		for(i=1;i<N;i++)
		{
			if(i%2==0)
				sum+=2*f(a+i*h);
			else
				sum+=4*f(a+i*h);
		}
		sum+=(f(a)+f(b));
		sum *=h/3;
		printf("\n%d\t\t|\t%d\t\t|\t%lf\t\t|\t%lf",k,N,h,sum);
		if(fabs(sum-old_sum)<epsilon)
		{
			printf("\n\n-->The Estimate of the Integral Using simpsons1/3 Rule is %lf",sum);
			break;
		}
		N*=2;
		k++;
```

```
		old_sum=sum;
	}
	printf("\n\n");
}

void simpsons3_8(double a,double b,int N)
{
	printf("=====================================Simpsons 3/8 Rule===================================\n\n");
	int i,limit=20,k=1;
	double sum=0,old_sum=0,h;
	printf("\nSr No\t\t|\tN\t\t|\th\t\t\t|\tIntegral\n");
	printf("__________________________________________________________________________________________");
	while(k<=limit)
	{
		sum=0;
		h=(b-a)/N;
		for(i=1;i<N;i++)
		{
			if(i%3==0)
				sum+=2*f(a+i*h);
			else
				sum+=3*f(a+i*h);
		}
		sum+=(f(a)+f(b));
		sum *=3*h/8;
		printf("\n%d\t\t|\t%d\t\t|\t%lf\t\t|\t%lf",k,N,h,sum);
		if(fabs(sum-old_sum)<epsilon)
		{
			printf("\n\n-->The Estimate of the Integral Using simpsons3/8 Rule is %lf",sum);
```

```
                    break;
                }
                N*=2;
                k++;
                old_sum=sum;
        }
}
```

********************************************************************************

output:

```
===================================Simpsons 1/3 Rule===================================

Sr No   |   N    |   h          |   Integral
_______________________________________________________________________________
1       |   2    |   1.000000   |   1.111111
2       |   4    |   0.500000   |   1.100000
3       |   8    |   0.250000   |   1.098725
4       |   16   |   0.125000   |   1.098620
5       |   32   |   0.062500   |   1.098613
6       |   64   |   0.031250   |   1.098612

-->The Estimate of the Integral Using simpsons1/3 Rule is 1.098612

===================================Simpsons 3/8 Rule===================================

Sr No   |   N    |   h          |   Integral
_______________________________________________________________________________
```

| | | | |
|---|---|---|---|
| 1 | 2 | 1.000000 | 1.062500 |
| 2 | 4 | 0.500000 | 1.056250 |
| 3 | 8 | 0.250000 | 1.087541 |
| 4 | 16 | 0.125000 | 1.087999 |
| 5 | 32 | 0.062500 | 1.095955 |
| 6 | 64 | 0.031250 | 1.095995 |
| 7 | 128 | 0.015625 | 1.097958 |
| 8 | 256 | 0.007813 | 1.097960 |
| 9 | 512 | 0.003906 | 1.098449 |
| 10 | 1024 | 0.001953 | 1.098449 |

-->The Estimate of the Integral Using simpsons3/8 Rule is 1.098449

********************************************************************************

*******************************************************************************

Q(3): A car laps a race track in 84 seconds. The speed of the car at each 6-second interval is determined by using a radar gun and is given from the beginning of the lap, in feet/second by the entries in the following table.

Time 0 6 12 18 24 30 36 42 48 54 60 66 72 78 84

Speed 124 134 148 156 147 133 121 109 99 85 78 89 104 116 123

How long is the track?

Use (i) Trapezoidal Rule (ii) Simpson's 1/3 rule (iii) Simpson's3/8 rule

*******************************************************************************

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

void simpsons1_3(double,double,int);
void simpsons3_8(double,double,int);
void trapezoidal(double,double,int);

double f(int x)
{
    switch(x)
    {
        case 0:return 124;
        case 6:return 134;
        case 12:return 148;
        case 18:return 156;
        case 24:return 147;
        case 30:return 133;
        case 36:return 121;
        case 42:return 109;
        case 48:return 99;
```

```
		case 54:return 85;
		case 60:return 78;
		case 66:return 89;
		case 72:return 104;
		case 78:return 116;
		case 84:return 123;
	}
}
void main()
{
	int N=14;
	double a,b;
	a=0;
	b=84;
	trapezoidal(a,b,N);
	simpsons1_3(a,b,N);
	simpsons3_8(a,b,N);
	getch();
}

void trapezoidal(double a,double b,int N)
{
	int i;
	double sum=0,h;
	sum=0;
	h=(b-a)/N;
	for(i=1;i<N;i++)
	{
		sum+=2*f(a+i*h);
	}
	sum+=(f(a)+f(b));
```

```
        sum *=h/2;
        printf("\n-->Length of track using Trapezoidal Rule=%0.2lf Feet",sum);
}

void simpsons1_3(double a,double b,int N)
{
        int i;
        double sum=0,h;
                sum=0;
                h=(b-a)/N;
                for(i=1;i<N;i++)
                {
                        if(i%2==0)
                                sum+=2*f(a+i*h);
                        else
                                sum+=4*f(a+i*h);
                }
                sum+=(f(a)+f(b));
                sum *=h/3;
                printf("\n-->Length of track using Simpsons 1/3 Rule=%0.2lf Feet",sum);
}

void simpsons3_8(double a,double b,int N)
{
        int i;
        double sum=0,h;
        sum=0;
        h=(b-a)/N;
        for(i=1;i<N;i++)
        {
                if(i%3==0)
```

```
                sum+=2*f(a+i*h);
            else
                sum+=3*f(a+i*h);
        }
        sum+=(f(a)+f(b));
        sum *=3*h/8;
        printf("\n-->Length of track using Simpsons 3/8 Rule=%0.2lf Feet",sum);
}
```

*******************************************************************************

output:

```
-->Length of track using Trapezoidal Rule=9855.00 Feet
-->Length of track using Simpsons 1/3 Rule=9858.00 Feet
-->Length of track using Simpsons 3/8 Rule=9760.50 Feet
```

*******************************************************************************

}

*********************************************************************************

Q(4): Write a program to solve the differential equation dy/dx=(y-x)/(y+x), where y(0) = 1, using

(i) Euler's method

(ii) Runge - Kutta second order method

in the interval 0 to 1 using step-size 0.1 Tabulate your results

*********************************************************************************

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void euler(double,double,double,int);
void runge_kutta_2(double,double,double,int);

double f(double x,double y)
{
	return ((y-x)/(y+x));
}
void main()
{
	int limit;
	double xi,yi,h;
	xi=0;
	yi=1;
	h=0.1;
	limit=1;
	euler(xi,yi,h,limit);
	runge_kutta_2(xi,yi,h,limit);
	getch();
}

void euler(double xi,double yi,double h,int limit)
```

```
{
	double yi_1;
	yi_1=yi;
	printf("=============EULER METHOD============\n\n");
	printf("\nx\t\t|\tSolution\n");
	printf("____________________________________\n");
	while(xi<=limit)
	{
		yi=yi_1;
		printf("\n%0.2lf\t\t|\t%lf",xi,yi);
		yi_1=yi + h* f(xi,yi);
		xi+=h;
	}
	printf("\n\n-->Solution With Eulers method= %lf\n\n",yi);
}

void runge_kutta_2(double xi,double yi,double h,int limit)
{
	double yi_1,k0,k1;
	yi_1=yi;
	printf("========RUNGE-KUTTA SECOND ORDER METHOD========\n\n");
	printf("\nx\t\t|\tSolution\n");
	printf("____________________________________");
	while(xi<=limit)
	{
		yi=yi_1;
		printf("\n%0.2lf\t\t|\t%lf",xi,yi);
		k0=h*f(xi,yi);
		k1=h*f(xi+h,yi+k0);
		yi_1=yi + (0.5)*(k0+k1);
		xi+=h;
```

```
    }
    printf("\n\n-->Solution With RUNGE-KUTTA SECOND ORDER METHOD= %lf",yi);
}
```

*******************************************************************************

output:

```
==============EULER METHOD============


x           |     Solution
__________________________________________


0.00        |     1.000000
0.10        |     1.100000
0.20        |     1.183333
0.30        |     1.254418
0.40        |     1.315818
0.50        |     1.369193
0.60        |     1.415694
0.70        |     1.456161
0.80        |     1.491231
0.90        |     1.521399
1.00        |     1.547062

-->Solution With Eulers method= 1.547062

========RUNGE-KUTTA SECOND ORDER METHOD=======


x           |     Solution
__________________________________________
```

| | |
|---|---|
| 0.00 | 1.000000 |
| 0.10 | 1.091667 |
| 0.20 | 1.168728 |
| 0.30 | 1.234629 |
| 0.40 | 1.291489 |
| 0.50 | 1.340729 |
| 0.60 | 1.383361 |
| 0.70 | 1.420135 |
| 0.80 | 1.451627 |
| 0.90 | 1.478291 |
| 1.00 | 1.500491 |

-->Solution With RUNGE-KUTTA SECOND ORDER METHOD= 1.500491

********************************************************************************

********************************************************************************

Q(5): Find the solution of differential equation, for the range 0 <= t <= 1 dy/dt = t + (y)^(1/2)

with y(0) = 1, taking step size h = 0.2 using Runge-Kutta method of order 4

********************************************************************************

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

void runge_kutta_4(double,double,double,int);

double f(double t,double y)
{
	return (t+sqrt(y));
}
void main()
{
	int limit;
	double ti,yi,h;
	ti=0;
	yi=1;
	h=0.2;
	limit=1;
	runge_kutta_4(ti,yi,h,limit);
	getch();
}
void runge_kutta_4(double ti,double yi,double h,int limit)
{
	double yi_1,k0,k1,k2,k3;
	yi_1=yi;
	printf("=======RUNGE-KUTTA FORTH ORDER METHOD=======\n\n");
```

```
	printf("\nt\t\t|\tSolution\n");
	printf("____________________________________");
	while(ti<=limit)
	{
		yi=yi_1;
		printf("\n%0.2lf\t\t|\t%lf",ti,yi);
		k0=h*f(ti,yi);
		k1=h*f(ti+(h/2),yi+(k0/2));
		k2=h*f(ti+(h/2),yi+(k1/2));
		k3=h*f(ti+h,yi+k2);
		yi_1=yi + (k0+2*k1+2*k2+k3)/6;
		ti+=h;
	}
	printf("\n\n-->Solution With RUNGE-KUTTA FORTH ORDER METHOD= %lf",yi);
}
```

*******************************************************************************

output:

```
========RUNGE-KUTTA FORTH ORDER METHOD========


t		|	Solution
____________________________________
0.00		|	1.000000
0.20		|	1.230632
0.40		|	1.524809
0.60		|	1.885413
0.80		|	2.314716
1.00		|	2.814506
-->Solution With RUNGE-KUTTA FORTH ORDER METHOD= 2.814506
```

*******************************************************************************

********************************************************************************

Q(6): Find the solution of differential equation dy/dt = 1/2 (t+y), for y (2.0) given

y(0) = 2

y(0.5) = 2.636

y(1.0) = 3.595

and y(1.5) = 4.968 , use h = 0.5

using (i) Milne-Simpson's predictor corrector method

(ii) Adam-Bashforth-Moulton's predictor-corrector method

********************************************************************************

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

#define epsilon 0.00005

void milne_simpsone_predictor_corrector(double[],double[],double);

void adam_bashforth_moultons_predictor_corrector(double[],double[],double);


double f(double t,double y)

{

	return ((t+y)/2);

}

void main()

{

	double h,y[10],t[10];

	h=0.5;

	y[0]=2;

	y[1]=2.636;

	y[2]=3.595;

	y[3]=4.968;

	t[0]=0;

	t[1]=0.5;
```

```
        t[2]=1.0;
        t[3]=1.5;
        t[4]=2.0;
        milne_simpsone_predictor_corrector(y,t,h);
        adam_bashforth_moultons_predictor_corrector(y,t,h);
        getch();
}
void milne_simpsone_predictor_corrector(double y[],double t[],double h)
{
        double yi_old=0;
        int i;
        i=3;
        printf("========milne_simpsone_predictor_corrector METHOD========\n\n");
        //predictor Method
        y[i+1]=y[i-3]+(4*h)*(2*f(t[i],y[i])-f(t[i-1],y[i-1])+2*f(t[i-2],y[i-2]))/3;
        printf("Using Predictor Formula y4 =%lf",y[i+1]);



        //Corrector formula
        while(fabs(yi_old-y[i+1])>epsilon)
        {
                yi_old=y[i+1];
                y[i+1]=y[i-1] + (h/3) *(f(t[i+1],y[i+1])+ 4* f(t[i],y[i])+f(t[i-1],y[i-1]));
                printf("\n-->Using Corrector Formula y4=%lf",y[i+1]);
        }
        printf("\n\n---->Solution With milne_simpsone_predictor_corrector METHOD=
%lf\n\n",y[i+1]);
}

void adam_bashforth_moultons_predictor_corrector(double y[],double t[],double h)
{
```

```
	double yi_old=0;
	int i;
	i=3;
	printf("========adam_bashforth_moultons_predictor_corrector METHOD========\n\n");
	//predictor Method
	y[i+1]=y[i]+(h/24)*(55*f(t[i],y[i])-59*f(t[i-1],y[i-1])+37*f(t[i-2],y[i-2])-9*f(t[i-3],y[i-3]));
	printf("Using Predictor Formula y4 =%lf",y[i+1]);



	//Corrector formula
	while(fabs(yi_old-y[i+1])>epsilon)
	{
		yi_old=y[i+1];
		y[i+1]=y[i] + (h/24) *(9*f(t[i+1],y[i+1])+ 19 * f(t[i],y[i])-5*f(t[i-1],y[i-1])+f(t[i-2],y[i-2]));
		printf("\n-->Using Corrector Formula y4=%lf",y[i+1]);
	}
	printf("\n\n---->Solution With adam_bashforth_moultons_predictor_corrector METHOD=
%lf",y[i+1]);
}
```

*********************************************************************************

output:

```
========milne_simpsone_predictor_corrector METHOD========

Using Predictor Formula y4 =6.871000
-->Using Corrector Formula y4=6.873167
-->Using Corrector Formula y4=6.873347
-->Using Corrector Formula y4=6.873362

---->Solution With milne_simpsone_predictor_corrector METHOD= 6.873362
```

========adam_bashforth_moultons_predictor_corrector METHOD========

Using Predictor Formula y4 =6.870781

-->Using Corrector Formula y4=6.873104

-->Using Corrector Formula y4=6.873322

-->Using Corrector Formula y4=6.873343

---->Solution With adam_bashforth_moultons_predictor_corrector METHOD= 6.873343

******************************************************************************************

********************************************************************************

Q(7): Use Adam-Bashforth-Moulton's predictor-corrector method to obtain the solution of the equation dy/dx= 1-xy/x^2 at x = 1.4, where y(1) = 1.

Compute y(1.1), y(1.2) and y(1.3) using Runge-Kutta second order method.

Tabulate the results obtained thus.

********************************************************************************

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define epsilon 0.00005

void adam_bashforth_moultons_predictor_corrector(double[],double[],double);
double runge_kutta_2(double,double,double,double);
double f(double x,double y)
{
	return ((1-x*y)/(x*x));
}
void main()
{
	double h,y[10],x[10];
	h=0.1;
	x[0]=1;
	x[1]=1.1;
	x[2]=1.2;
	x[3]=1.3;
	x[4]=1.4;

	y[0]=1;
	y[1]=runge_kutta_2(x[0],y[0],h,1.2);
	y[2]=runge_kutta_2(x[0],y[0],h,1.3);
```

```
            y[3]=runge_kutta_2(x[0],y[0],h,1.4);


            printf("\n========================By Runge-Kutta second order method\n");
            printf("y(1.1)=%lf\ny(1.2)=%lf\ny(1.3)=%lf\n\n",y[1],y[2],y[3]);
            adam_bashforth_moultons_predictor_corrector(y,x,h);
            getch();
}

void adam_bashforth_moultons_predictor_corrector(double y[],double x[],double h)
{
            double yi_old=0;
            int i;
            i=3;
            printf("========adam_bashforth_moultons_predictor_corrector METHOD========\n\n");
            //predictor Method
            y[i+1]=y[i]+(h/24)*(55*f(x[i],y[i])-59*f(x[i-1],y[i-1])+37*f(x[i-2],y[i-2])-9*f(x[i-3],y[i-3]));
            printf("Using Predictor Formula y(1.4) =%lf",y[i+1]);



            //Corrector formula
            while(fabs(yi_old-y[i+1])>epsilon)
            {
                        yi_old=y[i+1];
                        y[i+1]=y[i] + (h/24) *(9*f(x[i+1],y[i+1])+ 19 * f(x[i],y[i])-5*f(x[i-1],y[i-1])+f(x[i-2],y[i-
2]));
                        printf("\n-->Using Corrector Formula y(1.4)=%lf",y[i+1]);
            }
            printf("\n\n---->Solution With adam_bashforth_moultons_predictor_corrector METHOD=
%lf",y[i+1]);
}

double runge_kutta_2(double xi,double yi,double h,double limit)
```

```
{
	double yi_1,k0,k1;
	yi_1=yi;
	while(xi<limit)
	{
		yi=yi_1;
		k0=h*f(xi,yi);
		k1=h*f(xi+h,yi+k0);
		yi_1=yi + (0.5)*(k0+k1);
		xi+=h;
	}
	return yi;
}
```

```
**************************************************************************
output:
========================By Runge-Kutta second order method
y(1.1)=0.995868
y(1.2)=0.985480
y(1.3)=0.971311

========adam_bashforth_moultons_predictor_corrector METHOD========

Using Predictor Formula y(1.4) =0.954695
-->Using Corrector Formula y(1.4)=0.954878
-->Using Corrector Formula y(1.4)=0.954873
---->Solution With adam_bashforth_moultons_predictor_corrector METHOD= 0.954873
**************************************************************************
```

*******************************************************************************

Q(8): Use Milne Simpson predictor corrector method to obtain the solution of the equation dy/dx= 1-xy/x^2 at x = 1.4, where y(1) = 1.

Compute y(1.1), y(1.2) and y(1.3) using Runge-Kutta fourth order method.

Tabulate the results obtained thus.

*******************************************************************************

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define epsilon 0.00005

void milne_simpsone_predictor_corrector(double[],double[],double);
double runge_kutta_4(double,double,double,double);
double f(double x,double y)
{
	return ((1-x*y)/(x*x));
}
void main()
{
	double h,y[10],x[10];
	h=0.1;
	x[0]=1;
	x[1]=1.1;
	x[2]=1.2;
	x[3]=1.3;
	x[4]=1.4;

	y[0]=1;
	y[1]=runge_kutta_4(x[0],y[0],h,1.2);
	y[2]=runge_kutta_4(x[0],y[0],h,1.3);
```

```
	y[3]=runge_kutta_4(x[0],y[0],h,1.4);

	printf("\n========================By Runge-Kutta Forth order method\n");
	printf("y(1.1)=%lf\ny(1.2)=%lf\ny(1.3)=%lf\n\n",y[1],y[2],y[3]);
	milne_simpsone_predictor_corrector(y,x,h);
	getch();
}
void milne_simpsone_predictor_corrector(double y[],double x[],double h)
{
	double yi_old=0;
	int i;
	i=3;
	printf("========milne_simpsone_predictor_corrector METHOD========\n\n");
	//predictor Method
	y[i+1]=y[i-3]+(4*h)*(2*f(x[i],y[i])-f(x[i-1],y[i-1])+2*f(x[i-2],y[i-2]))/3;
	printf("Using Predictor Formula y(1.4)=%lf",y[i+1]);



	//Corrector formula
	while(fabs(yi_old-y[i+1])>epsilon)
	{
		yi_old=y[i+1];
		y[i+1]=y[i-1] + (h/3) *(f(x[i+1],y[i+1])+ 4* f(x[i],y[i])+f(x[i-1],y[i-1]));
		printf("\n-->Using Corrector Formula y(1.4)=%lf",y[i+1]);
	}
	printf("\n\n---->Solution With milne_simpsone_predictor_corrector METHOD=
%lf\n\n",y[i+1]);
}
double runge_kutta_4(double xi,double yi,double h,double limit)
{
	double yi_1,k0,k1,k2,k3;
```

```
        yi_1=yi;
        while(xi<limit)
        {
                yi=yi_1;
                k0=h*f(xi,yi);
                k1=h*f(xi+(h/2),yi+(k0/2));
                k2=h*f(xi+(h/2),yi+(k1/2));
                k3=h*f(xi+h,yi+k2);
                yi_1=yi + (k0+2*k1+2*k2+k3)/6;
                xi+=h;
        }
        return yi;
}
```

********************************************************************************

output:

```
========================By Runge-Kutta Forth order method
y(1.1)=0.995737
y(1.2)=0.985268
y(1.3)=0.971050

========milne_simpsone_predictor_corrector METHOD========

Using Predictor Formula y(1.4)=0.954478
-->Using Corrector Formula y(1.4)=0.954629
-->Using Corrector Formula y(1.4)=0.954626
---->Solution With milne_simpsone_predictor_corrector METHOD= 0.954626
```

********************************************************************************

*******************************************************************************

Q(9): From the following table estimate y'(1.1) and y'(1.2) using 3 point formulas and 5 point formulas

x 1.0 1.05 1.10 1.15 1.20 1.25 1.30

y 1.0 1.0247 1.0488 1.0724 1.0954 1.1180 1.1402

*******************************************************************************

```
#include<stdio.h>

#include<conio.h>

#include<math.h>


void _3point_formulas(double[],double[],double);

void _5point_formulas(double[],double[],double);


void main()

{

	double x[10],y[10],h=0.5;

	x[0]=1.0;

	x[1]=1.05;

	x[2]=1.10;

	x[3]=1.15;

	x[4]=1.20;

	x[5]=1.25;

	x[6]=1.30;

	y[0]=1.0;

	y[1]=1.0247;

	y[2]=1.0488;

	y[3]=1.0724;

	y[4]=1.0954;

	y[5]=1.1180;

	y[6]=1.1402;
```

```
	_3point_formulas(x,y,h);
	_5point_formulas(x,y,h);
	getch();
}

void _3point_formulas(double x[],double y[],double h)
{

	double x0=x[2],ans;
	int i=2;
	//Endpoint formula
	printf("\n===============3 Pont End Point Formula===============\n");
	ans=(1/(2*h)) * (-3 * y[i] + 4*y[i+1]-y[i+2]);
	printf("\n--->y(1.1)'=%lf",ans);
	i=4;
	ans=(1/(2*h)) * (-3 * y[i] + 4*y[i+1]-y[i+2]);
	printf("\n--->y(1.2)'=%lf",ans);
	//Midpoint Formula
	i=2;
	printf("\n===============3 Pont Mid Point Formula===============\n");
	ans=(1/(2*h)) * (-y[i-1] + y[i+1]);
	printf("\n--->y(1.1)'=%lf",ans);
	i=4;
	ans=(1/(2*h)) * (-y[i-1] + y[i+1]);
	printf("\n--->y(1.2)'=%lf",ans);
	//Endpoint formula
	printf("\n===============3 Pont End Point Formula===============\n");
	i=2;
	ans=(1/(2*h)) * (y[i-2] - 4*y[i-1]+3*y[i]);
	printf("\n--->y(1.1)'=%lf",ans);
```

```
        i=4;
        ans=(1/(2*h)) * (y[i-2] - 4*y[i-1]+3*y[i]);
        printf("\n--->y(1.2)'=%lf",ans);
}


void _5point_formulas(double x[],double y[],double h)
{

        double x0=x[2],ans;
        int i=2;
        //Endpoint formula
        printf("\n\n\n===============5 Pont End Point Formula===============\n");
        ans=(1/(12*h)) * ( -25*y[i] +48*y[i+1]-36* y[i+2]+16*y[i+3]-3*y[i+4]);
        printf("\n--->y(1.1)'=%lf",ans);

        //Midpoint Formula
        i=2;
        printf("\n===============5 Pont Mid Point Formula===============\n");
        ans=(1/(12*h)) * ( y[i-2] - 8*y[i-1]+8* y[i+1]-y[i+2]);
        printf("\n--->y(1.1)'=%lf",ans);
        i=4;
        ans=(1/(12*h)) * ( y[i-2] - 8*y[i-1]+8* y[i+1]-y[i+2]);
        printf("\n--->y(1.2)'=%lf",ans);
}
```

*******************************************************************************

output:

```
===============3 Pont End Point Formula===============
```

--->y(1.1)'=0.047800

--->y(1.2)'=0.045600

===============3 Pont Mid Point Formula===============

--->y(1.1)'=0.047700

--->y(1.2)'=0.045600

===============3 Pont End Point Formula===============

--->y(1.1)'=0.047600

--->y(1.2)'=0.045400

===============5 Pont End Point Formula===============

--->y(1.1)'=0.048033

===============5 Pont Mid Point Formula===============

--->y(1.1)'=0.047700

--->y(1.2)'=0.045567

*******************************************************************************