```
#include<bits/stdc++.h>
#include <math.h>
using namespace std;

int degree=5, itr = 0, r = 1;
double e, q[100],a[100], a0, a1;

double horner(double n)
{
   double p[100];
   p[degree] = q[degree];
   for(int i = degree; i > 0; i--)
   {
      p[i-1] = p[i]*n + q[i-1];
   }
   return p[0];
}

double primeHornor(double x)
{
   double p1[100], d[100];
   for(int i = 1; i <= degree; i++)
   {
      d[i-1] = q[i]*i;
   }
   p1[degree - 1] = d[degree- 1];
   for(int i = degree-1; i > 0; i--)
   {
      p1[i-1] = p1[i]*x + d[i-1];
```

```
    }
    return p1[0];
}

void polynomialDeflation(double x)
{
    q[degree] = 0;
    for(int i = degree; i > 0; i--)
    {
        q[i-1] = a[i] + q[i]*x;
    }
    for(int i = degree; i > -1; i--)
    {
        a[i] = q[i];
    }
    a0 = a[0];
    a1 = a[1];
}



void newton(double a){
    double f0, x0, fr, xr,fb,relat_error;
    x0 = a;
    f0 = horner(x0);
    int order=degree;
    if(primeHornor(x0) == 0)
    {
        return;
    }
```

```
  while(degree > 1)
  {
     while(true)
     {
        f0 = horner(x0);
        fb = primeHornor(x0);

        if(fb == 0)
        {
           break;
        }
        else
        {itr++;
           xr = x0 - (f0/fb);
           if(f0 == 0)
           {
              return;
           }
           relat_error=fabs((xr-x0)/xr);
           x0 = xr;

           if(fabs(horner(xr)) < 0.001)
           {
              printf("\nAt the order %d the Root is %lf after %d iteration and relative error %lf",order,xr
,itr,relat_error);
              order--;r++;
              printf("\nThere Root is close to the real Root\n");
              break;
```

```
            }
            else
            {
               x0 = xr;
            }

         }
       }
   itr++;
      polynomialDeflation(xr);
      degree--;
   }
   double root = -a0/a1;
   relat_error=fabs((xr-x0)/xr);
   printf("\nAt the order %d the Root is %lf after %d iteration and relative error %lf",order,root
,itr,relat_error);
   if(fabs(horner(root))==0){
         printf("\nThere Root is not real Root\n");}
   printf("There are %d Roots for the given polynomial\n",r);
}
double MaxRoot(){
  return -a[5-1]/a[5];
}
int main()
{
   double root,max_root;

   cout<<"Enter the value of coefficients: "<<endl;
   for(int i = degree; i > -1; i--)
```

```
    {
        printf("Coefficients x[%d] = ", i);
        cin>>a[i];
        q[i] = a[i];
    }
    cout<<endl;
    max_root=MaxRoot();
    printf("\nLargest possible root is  %lf\n",max_root);
    newton(max_root);

    return 0;
}
```