```
//Question 1-------------------------------------------------------------------------------
-----

/*
grow a tree

*/

#include <iostream>
#include <cstdlib>
#include <fstream>


using std::endl;
using std::cout;
using std::ofstream;

int main()
{
   int i;
   double x, y, xn, yn, r;

  // setup file i/o:

   ofstream out_file;
   const char out_file_name[] = "tree.dat";  // save data in fern.dat

   out_file.open(out_file_name);

   x     = 0.5;                          // starting point
   y     = 0.0;

   const int max = 100000;    // number of iterations
   srand((time(0)));         // seed random nmbr generator.
                             // srand & time built-in

   for(i=1; i<=max; ++i)     // iterations
     {
     r = rand()/(double)RAND_MAX;       // scale random nmbr

     if (r <= 0.1)                      // stem
       {
         xn = 0.05*x;
         yn = 0.6*y;
       }
      else if((r>0.1) && (r<=0.2))      // right leaf
        {
        xn = 0.05*x;
        yn = -0.5*y+1;
        }
      else if ((r>0.2) && (r<=0.4))     // left leaf
        {
        xn = 0.46*x  - 0.15*y;
        yn = 0.39*x + 0.38*y + 0.6;
        }
      else if ((r>0.4) && (r<=0.6))                              // shoot
        {
        xn = 0.47*x - 0.15*y ;
        yn = 0.17*x + 0.42*y + 1.1;
        }
      else if ((r>0.6) && (r<=0.8))
        {
        xn = 0.43*x + 0.28*y ;
```

```
        yn = -0.25*x + 0.45*y + 1;
        }
      else
        {
        xn = 0.42*x + 0.26*y ;
        yn = -0.35*x + 0.31*y + 0.7;
        }

     x=xn;
     y=yn;
     out_file << x << " " << y << endl; // write date to output file
     }
   out_file.close();
   cout << "data stored in tree.dat" << endl;

   return 0;
}

//use gnuplot to plot the tree.dat


//Question 2-------------------------------------------------------------------------------
-----


/*
newton-raphson method for root finding.

*/

#include <iostream>
#include <cstdlib>
#include <cmath>
#include <fstream>

using std::endl;
using std::cout;
using std::ofstream;

// define function:

double func_y(double x){
  return sqrt(pow(x,2)+2*x+1)-2*x*sin(x); /* <---- for 2nd function , replace this by log10(
x), or tanh(x) for the 1st function . */
}


//define first derivative of function:

double func_dydx(double s){
  return (s+1)/sqrt(pow(s,2)+2*s+1)-2*sin(s)-2*s*cos(s); /* <---- for the 2nd function, repl
ace this by 1/s, or pow(cosh(s),-2) for the 3rd function. */
}


int main()
{
   int N= 0;
   int N_limit = 1000;           // max number of trys
   double s = 2.0;              // initial guess for root, /* <---- for the 2nd function, re
place this by 3, or -5 for the 3rd function. */
   double tol = 1.0e-7;          // tolerance for finding zero
```

```
   ofstream out_file;
   const char out_file_name[] = "newton.dat";  // save data in fern.dat

   out_file.open(out_file_name);

   while ((fabs(func_y(s)) >= tol) && (N < N_limit)){

        //cout << " " << func_y(s) << " " << func_dydx(s) << " " << " " << func_y(s)/func_dy
dx(s) << s << N << endl;
     s = s - func_y(s)/func_dydx(s);
     ++N;

   out_file << N << " " << s << endl; // write date to output file

        }

   if(N >= N_limit){
     cout << "iteration limit exceeded" << endl;
     return 0;
   }

   if (N <= N_limit){
     cout << "estimated root = " << s << " with " << N << " iterations." << endl;
   }

   return 0;
}


// for the first function, since 1/cosh(x) = 0,  Newton-Raphson Technique can not find the r
oot. so use Bisection Algorithm:

/*
bisection method for root finding
of function f(x) in closed interval [a,b].
*/

#include <iostream>
#include <cstdlib>
#include <cmath>
#include <fstream>

using std::endl;
using std::cout;
using std::ofstream;

// define function:

double func_f(double x){
  return tanh(x);
}


int main()
{
   int N= 0;
   int N_limit = 1000;           // max number of trys

   double a = -10.0;   // lower bound for [a,b]
   double b = 3.0;      // upper bound for [a,b]

   double tol = 1.0e-10;          // tolerance for finding zero
```

```
   ofstream out_file;
   const char out_file_name[] = "Bisection.dat";  // save data in fern.dat

   out_file.open(out_file_name);

   if (func_f(a) == 0.0) {
     cout << a << " is a root." << endl;
     return 0;
   }

   if (func_f(b) == 0.0) {
     cout << b << " is a root." << endl;
     return 0;
   }

   if (func_f(a)*func_f(b) > 0.0){
     cout << "[" << a << "," << b << "] do not bracket a root." << endl;
     return 0;
   }

   double x_low = a;
   double x_high = b;
   double x_mid;

   while ((fabs(x_high - x_low) >= tol) && (N < N_limit)){
     x_mid = 0.5*(x_low + x_high);
     if ( func_f(x_mid)*func_f(x_high) < 0){
       x_low = x_mid;
     }
     else {
       x_high = x_mid;
     }
   out_file << x_mid << " " << N << endl; // write date to output file
     ++N;
   }

   if(N > N_limit){
     cout << "iteration limit exceeded" << endl;
     return 0;
   }

   if (N <= N_limit){
     cout << "estimated root = " << x_mid << " with " << N << " iterations." << endl;
   }

   return 0;
}

//use gnuplot to make plots from the .dat files.


//Question 3-------------------------------------------------------------------------
-----

//for Newton-Raphson Technique, the source code is:


/*
newton-raphson method for root finding.

*/
```

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <fstream>

using std::endl;
using std::cout;
using std::ofstream;

// define function:

double func_y(double x){
  return sin(x); /* <---- for 2nd function , replace this by x*sqrt(fabs(x)) */
}


//define first derivative of function:

double func_dydx(double s){
  return cos(s); /* <---- for the 2nd function, replace this by sqrt(fabs(s))+0.5*pow(s,2)*p
ow(sqrt(fabs(s)),1.5) */
}


int main()
{
   int N= 0;
   int N_limit = 1000;           // max number of trys
   double s = 0.1;              // initial guess for root, /* <---- for the 2nd function, re
place this by 3, or -5 for the 3rd function. */
   double tol = 1.0e-4;          // tolerance for finding zero


   ofstream out_file;
   const char out_file_name[] = "newton.dat";  // save data in fern.dat

   out_file.open(out_file_name);

   while ((fabs(func_y(s)) >= tol) && (N < N_limit)){

     s = s - func_y(s)/func_dydx(s);
     ++N;

   out_file << N << " " << s << endl; // write date to output file

        }

   if(N >= N_limit){
     cout << "iteration limit exceeded" << endl;
     return 0;
   }

   if (N <= N_limit){
     cout << "estimated root = " << s << " with " << N << " iterations." << endl;
   }

   return 0;
}


//and for Bisection Algorithm, the source code is:

/*
```

```
bisection method for root finding
of function f(x) in closed interval [a,b].
*/

#include <iostream>
#include <cstdlib>
#include <cmath>
#include <fstream>

using std::endl;
using std::cout;
using std::ofstream;

// define function:

double func_f(double x){
  return sin(x); // <----replace this with x*sqrt(fabs(x)) for the 2nd function.
}


int main()
{
   int N= 0;
   int N_limit = 1000;           // max number of trys

   double a = -2.9;   // lower bound for [a,b]
   double b = 3.0;      // upper bound for [a,b]

   double tol = 1.0e-4;          // tolerance for finding zero

   ofstream out_file;
   const char out_file_name[] = "Bisection.dat";  // save data in fern.dat

   out_file.open(out_file_name);

   if (func_f(a) == 0.0) {
     cout << a << " is a root." << endl;
     return 0;
   }

   if (func_f(b) == 0.0) {
     cout << b << " is a root." << endl;
     return 0;
   }

   if (func_f(a)*func_f(b) > 0.0){
     cout << "[" << a << "," << b << "] do not bracket a root." << endl;
     return 0;
   }

   double x_low = a;
   double x_high = b;
   double x_mid;

   while ((fabs(x_high - x_low) >= tol) && (N < N_limit)){
     x_mid = 0.5*(x_low + x_high);
     if ( func_f(x_mid)*func_f(x_high) < 0){
       x_low = x_mid;
     }
     else {
       x_high = x_mid;
     }
   out_file << x_mid << " " << N << endl; // write date to output file
```

```
     ++N;
   }

   if(N > N_limit){
     cout << "iteration limit exceeded" << endl;
     return 0;
   }

   if (N <= N_limit){
     cout << "estimated root = " << x_mid << " with " << N << " iterations." << endl;
   }

   return 0;
}

/*
iterations table:


                                        newton          Bisection

sin(x)                          2                       16

x*sqrt(fabs(x))         1                       16


*/

//Question 4-------------------------------------------------------------------------------
-----


/*
newton-raphson method for root finding.

*/

#include <iostream>
#include <cstdlib>
#include <cmath>
#include <fstream>

using std::endl;
using std::cout;
using std::ofstream;

// define function:

double func_y(double x){
  return x*tan(x)-sqrt(16.8-pow(x,2));
}


//define first derivative of function:

double func_dydx(double s){
  return tan(s)+s/pow(cos(s),2)+s/sqrt(16.8-pow(s,2));
}


int main()
{
   int N= 0;
```

```
   int N_limit = 1000;          // max number of trys
   double s = 2;              // initial guess for root, depth of potential well in MeV
   double tol = 1.0e-5;         // tolerance for finding zero


   ofstream out_file;
   const char out_file_name[] = "eigenvalue.dat";  // save data in eigenvalue.dat

   out_file.open(out_file_name);

   while ((fabs(func_y(s)) >= tol) && (N < N_limit)){

     s = s - func_y(s)/func_dydx(s);
     ++N;

   out_file << N << " " << s << endl; // write date to output file
      cout << N << " " <<  func_y(s) << " " <<  func_dydx(s) << " " << s << endl;
        }

   if(N >= N_limit){
     cout << "iteration limit exceeded" << endl;
     return 0;
   }

   if (N <= N_limit){
     cout << "estimated root = " << pow(s,2)/(0.483*4)-83 << " with " << N << " iterations."
 << endl; //pow(s,2)/(0.483*4)-83 is the energy presented by s
   }

   return 0;
}

/*


1 -3.98521 3.73443 2.80096
2 2.08192 10.6651 3.86812
3 0.339257 7.54803 3.67291
4 0.0109096 7.07306 3.62796
5 1.17462e-05 7.05784 3.62642
6 1.36344e-11 7.05782 3.62642
estimated root = -76.1931 with 6 iterations.


*/
```