Programming assignment #1

ENG 101 Winter '05

Due Friday Jan 28th at 11:59pm

For this assignment, you are asked to write four functions. One function involves only doubles, one, one involves arrays, and two of the functions involve a simple structure. In each case a sketch of an algorithm is provided. In many cases, code which implements similar algorithms have been discussed in class.

You will turn it in by creating a directory named "P1" in your "eng101" (note the lower case) directory. You will turn in two files. One named "p1functions.cc" and the other named "p1main.cc". Your main can simply be a copy of the one from the website. Your p1functions.cc file must not have a main in it. ***Your p1functions.cc must work with the p1main.cc found on the web!*** Be sure to test it right before you hand it in. The dist() function is worth 30 points, the next two are worth 35 points.

Finally, the fourth function listed, qroot(), will not be counted for points (though we will check to see if it works). It is entirely optional. However, we strongly recommend you at least try it. With some effort, you all should be able to do it.

`double dist(double acceleration, double time)`

Description:

This function is to compute distance traveled from a fixed point by an initially stationary object when a constant acceleration is applied. You may assume that time is non-negative.

Algorithm:

Recall that (when initially stationary) the distance traveled by an object when a constant acceleration is applied is distance =0.5* acceleration*time^2.

Notes:

Remember a^2=a*a. (And yes this one is as simple as it looks)

Sample results:

dist(2.0,1.0) returns 1.0
dist(1.0,2.0) returns 2.0

---

double listMax(double list[], int numElem)

Description:

This function takes an array of doubles and returns the largest value from the array. "numElem" is the number of elements in the array. You may assume at least one value is non-negative.

Algorithm:

Set some temporary double to be some negative value. Now walk through each element of the array, comparing the temporary value to it. If that temporary value is less than that element of the array, set the temporary value to have the value stored in that array element. Otherwise do nothing. Continue going through all of the elements of the array. Once you have gone though all of the elements of the array, the temporary value must be the largest element in the array.

Notes:

Be sure not to access more array elements then you have.

Sample results:

If A[5]={3.0, 4.0 ,-1.5, 4.2 ,5.0 } then list_max(A,5) will return the value 5.0, but list_max(A,4) would return the value 4.2.

For the next two functions, the structure you will use is:

```
struct complex
{
      double real;
      double img;
};
```

complex complex_sqrt(double value)

Description:

This function takes a (non-complex) input and returns the complex value which results.

Algorithm:

If the number is non-negative, set the real component equal to the square root of the value and the imaginary component equal to zero.

If the number is negative, change the sign (can be done by multiplying by –1) and take the square root of that number. Set the imaginary component to be that value and the real component to be zero.

Notes:

As we've seen in class, the function sqrt() is built-in to C++. It takes a single "double" argument and returns a double. You will need to use it.

Sample results:

a=complex_sqrt(4), where a is a struct of type "complex" will result in `a.real=2` and `a.img=0`. a=complex_sqrt(-4) will result in `a.real=0` and `a.img=2`

(See directions above. This function will not be counted for points—it is purely optional)

`complex qroot (double a, double b, double c, int which)`

Description:

This function computes one of two roots of a quadratic. If "which" is zero returns $\frac{-b+\sqrt{b^2-4ac}}{2a}$. If "which" is one it returns $\frac{-b-\sqrt{b^2-4ac}}{2a}$. You may assume which will be either zero or one and that "a" will never be zero. Your function should work for all other inputs.

Algorithm:

The important part here is that when you add a complex number to a real number only the real part is added. For example (3+2i) +5 = (5+2i). When you divide a complex number by a real number you divide both parts of the complex number. So (4+2i)/2=(2+i). You will almost certainly wish to use your complex_sqrt() function from above.

Notes:

Recall that we computed the quadratic roots in day 3 (see the webpage if you don't have the handout) but only if the roots were real.

Sample:

a=qroot(1.0, 0.0, -1.0, 0) would return a.real=1 and a.img=0
a=qroot(1.0, 0.0, 1.0, 1) would return a.real=0 and a.img=-1
