CS 1713, Introduction to Computer Science
Assignment 2, Spring 1998
Modify the Copy Program
Due Thursday, January 29, 1998

The assignment: In this assignment you will modify the quadratic equations program of Assignment 1. You will also practice creating a directory, copying and modifying a file, and using a makefile.

Details:

1. First create a new directory named "`assign2`"to hold the new assignment, and switch to that directory.

```
runner% mkdir assign2                    (create a new directory)
runner% cd assign2                   (change to the new directory)
```

   The next time you login and want to work inside this directory, you don't need to create it again, but you will need to execute the "`cd assign2`" command.

2. Copy the old "`roots.c`" source file into the new directory, using a new name "`roots2.c`":

```
runner% cd ..              (change back to your home directory)
runner% cp roots.c assign2/roots2.c          (copy the source)
runner% cd assign2                   (change to the new directory)
```

   Now once again you are in the `assign2` directory and can work on the new source file `roots2.c` (still the same as the old one). (We could have used the same name `roots.c` for the new file, since it is in a different directory from the old, but it is probably less confusing to use a new name.)

3. Make four changes to the program, using the `vi` editor to change the file `roots2.c`:

   (a) Add an `fprintf` statement so that the first thing the program prints will be:

```
Assignment 2: Program written by <Your full name>.
```

   Here `<Your full name>` should be your actual name.

   (b) Have the program handle the case of a degenerate quadratic equation (in case `a2` is `0`) by having it print an appropriate message followed by the one root, and then exit. You may assume that `a1` is not zero in this case.

   (c) Have the program handle the case of a quadratic with a single repeated root (in case `discriminant` is `0`) by having it print an appropriate message followed by the one repeated root, and then exit.

   (d) Have the program handle complex roots by having it print two lines in the form:

```
Root1. Real part: <xxx>, imaginary part:  <yyy>.
Root2. Real part: <xxx>, imaginary part: -<yyy>.
```

   (Here `<xxx>` and `<yyy>` should be the actual real and imaginary parts. Let d be the discriminant: $d = a_1^2 - 4\, a_0\, a_2$. In case $d < 0$, that is, a complex root, then the roots have real and imaginary parts given by

   real part $= (-a_1)/(2\, a_2)$, imaginary parts $= \pm\sqrt{(-d)}/(2\, a_2)$)

4. When you are ready to use the `lint` and `cc` commands to check and compile this program, you would use the following, similar to the copy program:

```
runner% lint -m -u roots2.c -lm
runner% cc -g -o roots2 roots2.c -lm
```

It is easy to mistype one or both of these commands, and you need to enter them over and over again as you debug your program (returning to the `vi` editor to make changes). For this reason it would be nice to store the commands somewhere so that you only need to enter them once carefully. Unix provides the `make` command for this purpose. First create a file named "`makefile`" with the following contents:

```
roots2:  roots2.c
        cc -g -o roots2 roots2.c -lm
lint: roots2.c
        lint -m -u roots2.c -lm
```

In this file there *must* be a single TAB character before the "`cc`" and before the final line starting with "`lint`". (You get a TAB just by hitting the key labeled "Tab" in the `vi` editor. You don't see this TAB character--it just looks like 8 blanks in the file.) Now if you type the command

```
runner% make lint
```

the system will look for a file named "`makefile`" and in this case will use the fourth line to carry out the "`lint`" command there.

To compile the program, you type the command

```
runner% make
```

and the system will look for a file named "`makefile`" and will carry out the "`cc`" command on the second line. (In case there have been no changes to the "`roots2.c`" file since the last time it was compiled, `make` will do nothing.) Having typed the file "`makefile`" carefully, there is no more chance of forgetting to type in "`-lm`" or some other mistake.

5. As before, use typescript to produce a listing to hand in. You should try input numbers that will exercise the new parts of this program. Here is data that you should include:

```
runner% roots2 0.0 2.0 4.0
Assignment 2: Program written by Neal R. Wagner.
Solving 0.000 x^2 + 2.000 x + 4.000 = 0
This is a linear equation (no quadratic term).
Root: -2.000
runner% roots2 1.0 -4.0 4.0
Assignment 2: Program written by Neal R. Wagner.
Solving 1.000 x^2 + -4.000 x + 4.000 = 0
There is a single repeated root.
Root: 2.000
runner% roots2 1.0 -4.0 -12.0
Assignment 2: Program written by Neal R. Wagner.
Solving 1.000 x^2 + -4.000 x + -12.000 = 0
Roots: 6.000, -2.000
runner% roots2 1.0 -6.0 25.0
Assignment 2: Program written by Neal R. Wagner.
Solving 1.000 x^2 + -6.000 x + 25.000 = 0
Roots are complex conjugates.
Root1.  Real part: 3.000, imaginary part:  4.000.
Root2.  Real part: 3.000, imaginary part: -4.000.
runner%
```