

Verifying Code

In class you looked at an Alice program to determine what problems it had. This is *white box testing* since you could look at the program and not just see what it does. In this exercise you will do *black box testing* in which you cannot see the code involved. This is also called *functional testing* since you test how the code functions without worrying about how it does the work (a type of abstraction).

Beginning programmers often use black box testing as the only method of testing (or used the most). As you will hopefully see, it is not trivial to do and it is easy to miss mistakes. Running code is an important way to see if it runs correctly. However, for a given amount of time spent, programmers tend to find more problems by carefully reading the code and thinking about its correctness (without running it). There are a number of techniques available for testing code.

In this exercise you will run code that is designed to tell you how to make change using the minimum number of coins. The user inputs the amount spent (between 1 cent and 100 cents) and is told the number of quarters, dimes, nickels and pennies to give back so the minimum number of coins is used.

Your job is to first come up with test cases to see if the program is correct. You don't want to try all 100 possibilities if you can avoid that. Thus, think about what would make cases unique. Write down your ideas for test cases and why they are important. Next, try your cases on each program to find out where it makes mistakes. There are 8 versions of the program. Each version has a unique bug. For most of the bugs, the code makes mistakes for several values that are related in the result given. For the other ones, they are for values that might be special: for example the values involved are at the limit of an if/else statement. If you don't find the problem by using the test cases you planned, try to think up new test cases and add them to your list to test. Work in pairs to help figure out any problems. As you find the problem with a code, test a few extra cases to see if you can determine exactly what the problem is (you may need to expand beyond your original set for this since you are carefully testing a specific problem multiple times). Then write down what you think is wrong with this version. In the end you can compare to what other groups thought to see if you agree.

To make sure you cannot look at the code, it was written in Java and then converted to byte code (in a jar file). This is code that Java systems can run but you cannot see the original source that created the code. This is the type of code you run in a web browser when you run a Java application. Here is how to run the program:

1. Copy all the versions of the code into a directory on your desktop.
2. Open up a terminal to run the program.
 - On a PC do: Start->All Programs->Accessories->Command Prompt.
 - On a Mac go to the directory Applications:Utility and double click on Terminal.app
3. Go to the directory where the programs are located
 - cd Desktop
 - cd "directory with files"

You can see what files are located where you are by doing "dir" on a PC or "ls" on a Mac. You should see the programs when you are in the correct place.

4. Run the program with

```
java -jar change1.jar
```

replace “change1.jar” with “change2.jar”, etc., as you want to test other versions of the program.

Try doing this with one program to make sure you can correctly run them. Once you do, running it again or with other programs is easy. Note you can use the “up arrow” to go to previous lines you typed in. This will make it very easy to repeat the program again. You can edit a line with the left/right arrows, the delete key, and typing in new stuff. If you have any questions about this, ask your peer leader.

Some details on how the code works

The problem is solved using a *greedy algorithm*. They have this name since they choose the largest thing they can do (are greedy). In this problem, the size goes from quarter to dime to nickel to penny. Thus, you look at how much change you must give. If you can give quarters (largest item), then do that. After you take the quarters given into account, give as many dimes as you can (the next largest item). You continue this way through nickels and pennies.

For example if you want to give change of 57 cents, you first give 2 quarters leaving 7 cents. You cannot give any dimes so you give 1 nickel leaving 2 cents. Then you give the 2 pennies. This gives 2 quarters, 1 nickel and 2 pennies, which is correct.

The greedy algorithm does not always work. As an exercise, try to come up with a different set of coins where the algorithm gives the wrong answer. Hint: the number of coins is not as important as their value.

Boolean conditions

Play a game using the decks of yellow and blue cards. Each blue card has an English sentence that is either true or false. Each yellow card has one of three logical connectors: not, and, or. The game is played as follows:

- Each player starts with 6 cards, 3 yellow and 3 blue.
- When it's your turn, if you can make a logical formula that evaluates to true using at least 4 of your cards, put down the formula and draw new cards. Otherwise, trade in one of your cards for a card of the same color.
- The first person to put down at least 12 cards wins.

You can group cards to show which one applies first. Place a space to indicate groups. By default, logical operators are done from left to right.

If you don't know if a statement is true or false, you can ask someone to find out.