

Verifying Code

An important part of creating software in computer science is making sure it runs correctly. In class you have been creating Alice programs and looking at the code. By seeing the code you can often find mistakes. 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 if they pay with a dollar bill so the minimum number of coins is returned.

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 (mistake). 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 some check. 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 small groups 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 others 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. It is important that you leave the programs in the "becs" directory. In the default setup, the files will be in a directory named "change" with a subdirectory "becs".
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"

By default, you should have a directory “change” on your desktop and thus “cd change”. You can see what files are located where you are by doing “dir” on a PC or “ls” on a Mac. For example, “dir” shows the directory “becs”. Doing “dir becs” shows the programs change1.class, etc.

4. Run the program with
java becs/Change1
replace “Change1” with “Change2”, 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. The program “change” is the correct version in case you want to see a version without mistakes.

Some details on how the code works

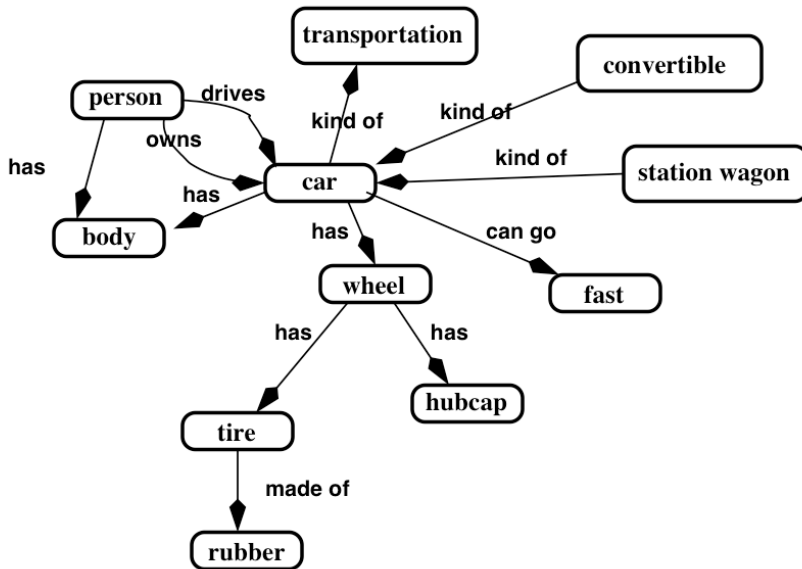
The problem is solved using a *greedy algorithm*. They have this name since they choose the largest thing they can do (are greedy). In this problem, the size goes from quarter to dime to nickel to penny. Thus, you look at how much change you must give. If you can give quarters (largest item), then do 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.

Concept Map

A *concept map* is a diagram that includes a set of *concepts* (or terms) and the *relationships* that connect them. Here's one example of a concept map:



Now let's try a concept map for the ideas so far in class.

1. Choose a *scribe* (someone to do the writing on the board/flip chart).
2. Create a set terms that you think are important from class so far. You may find it useful to look at the "Summary" and "Important concepts in the chapter" in Learning to Program with Alice.
3. Now work in groups to create your own concept map on a sheet of paper/flipchart. In each group, think of the relationships between the concepts on the board/flip chart. Each time you think of a relationship, write the concepts on your sheet of paper (if they're not already there), put an arrow between them, and label the arrow with the relationship you thought of. If there are concepts that no one in your group understands, look them up in the book or ask other people in the room. You may find you want to add some terms to the original list – that is fine.
4. When the groups finish, you can compare the results and discuss anything that interests people. Note that the exact words of the relationship aren't as important as what is related and how.

Saginaw Valley State University

Learning-Style Survey

For each question, provide a number from 1 to 4, where 1 means "Not at all true for me", and 4 means "Very true for me."

- (1) Making things for my studies helps me to remember what I have learned: ____
- (2) I can write about most of the things I know better than I can tell about them: ____
- (3) When I really want to understand what I'm reading, I read it softly to myself: ____
- (4) I get more done when I work alone: ____
- (5) I remember what I've read better than what I've heard: ____
- (6) When I answer questions, I can say the answer better than I can write it: ____
- (7) When I do math problems in my head, I say the numbers to myself: ____
- (8) I enjoy joining in on class discussions: ____
- (9) I understand a math problem that is written down better than one that I hear: ____
- (10) I do better when I can write the answer instead of having to say it: ____
- (11) I understand spoken directions better than written ones: ____
- (12) I like to work by myself: ____
- (13) I would rather read a story than listen to it read: ____
- (14) I would rather show and explain how something works than write about how it works: ____
- (15) If someone tells me three numbers to add, I can usually get the right answer without writing them down: ____
- (16) I prefer to work with a group when there is work to be done: ____
- (17) A graph or chart of numbers is easier for me to understand than hearing the numbers said: ____
- (18) Writing a spelling word several times helps me to remember it better: ____
- (19) I learn better if someone reads a book to me than if I read it silently to myself: ____
- (20) I learn best when I study alone: ____
- (21) When I have a choice between reading and listening, I usually read: ____
- (22) I would rather tell a story than write it: ____
- (23) Saying the multiplication tables over and over helps me remember them better than writing them over and over: ____
- (24) I do my best work in a group: ____
- (25) I understand a math problem that is written down better than one I hear: ____
- (26) In a group project, I would rather make a chart or poster than gather the information to put on it: ____
- (27) Written assignments are easy for me to follow: ____
- (28) I remember more of what I learn if I learn it alone: ____
- (29) I do well in classes where most of the information has to be read: ____
- (30) I would enjoy giving an oral report to the class: ____

- (31) I learn math better from spoken explanations than written ones: ____
- (32) If I have to decide something, I ask other people for their opinions: ____
- (33) Written math problems are easier for me to do than oral ones: ____
- (34) I like to make things with my hands: ____
- (35) I don't mind doing written assignments: ____
- (36) I remember things I hear better than things I read: ____
- (37) I learn better by reading than by listening: ____
- (38) It is easy for me to tell about the things I know: ____
- (39) I make it easier when I say the numbers of a problem to myself as I work it out: ____
- (40) If I understand a problem, I like to help someone else understand it, too: ____
- (41) Seeing a number makes more sense to me than hearing a number: ____
- (42) I understand what I have learned better when I am involved in making something for the subject:

- (43) The things I write on paper sound better than when I say them: ____
- (44) I find it easier to remember what I have heard than what I have read: ____
- (45) It is fun to learn with classmates, but it is hard to study with them: ____

SCORING

<p>Visual Language</p> <p>5 ____</p> <p>13 ____</p> <p>21 ____</p> <p>29 ____</p> <p>37 ____</p> <p>Total ____ x 2 = ____</p>	<p>Individual Learner</p> <p>4 ____</p> <p>12 ____</p> <p>20 ____</p> <p>28 ____</p> <p>45 ____</p> <p>Total ____ x 2 = ____</p>	<p>Auditory Numerical</p> <p>7 ____</p> <p>15 ____</p> <p>23 ____</p> <p>31 ____</p> <p>39 ____</p> <p>Total ____ x 2 = ____</p>
<p>Visual Numerical</p> <p>9 ____</p> <p>17 ____</p> <p>25 ____</p> <p>33 ____</p> <p>41 ____</p> <p>Total ____ x 2 = ____</p>	<p>Group Learner</p> <p>8 ____</p> <p>16 ____</p> <p>24 ____</p> <p>32 ____</p> <p>40 ____</p> <p>Total ____ x 2 = ____</p>	<p>Kinesthetic – Tactile</p> <p>1 ____</p> <p>18 ____</p> <p>26 ____</p> <p>34 ____</p> <p>42 ____</p> <p>Total ____ x 2 = ____</p>
<p>Auditory Language</p> <p>3 ____</p> <p>11 ____</p> <p>19 ____</p> <p>36 ____</p> <p>44 ____</p> <p>Total ____ x 2 = ____</p>	<p>Expressiveness – Oral</p> <p>6 ____</p> <p>14 ____</p> <p>22 ____</p> <p>30 ____</p> <p>38 ____</p> <p>Total ____ x 2 = ____</p>	<p>Expressiveness – Written</p> <p>2 ____</p> <p>10 ____</p> <p>27 ____</p> <p>35 ____</p> <p>43 ____</p> <p>Total ____ x 2 = ____</p>
<p style="text-align: center;">SCORE:</p> <p style="text-align: center;">33–40 = Major learning style</p> <p style="text-align: center;">20–32 = Minor learning style</p> <p style="text-align: center;">0 – 20 = Negligible use</p>		