CS 246 Fall 2015 - Tutorial 2

September 29, 2015

1 Summary

- Shell Scripting
- Testing Example
- Compiling Your First (C++) Program

2 Shell Scripting

2.1 Basic Scripting

- Command pipelines are great and we can do interesting things with them
- However, sometimes we need to do something more meaningful and this is where shell scripts come in
- The following is a simple shell script that may be familiar

```
1 #!/bin/bash
2 echo "Hello $(whoami)!"
3 echo "Today is $(date)."
```

- Suppose that this script is contained in the file "simple.script". How do we run this script?
- Answer 1: Use chmod to set the user executable bit
- Answer 2: Invoke bash with the script as a parameter

2.2 Command Line Arguments

- To make more complex shell scripts, we likely want to accept arguments from the command line
- Recall the following special shell variables:
  - `${#}` provides the number of arguments, excluding $0
  - `${0}` **always** contains the name of the script
  - `${1}, ${2}, ${3},...` refer to the arguments by position (not name)
  - `${@}` all arguments supplied to the currently running script (except $0), as separate strings
- Let's very quickly see these in action - see paramExample.script

2.3 More Complex Scripting

- Recall from lecture that bash shell scripts can have if-statements, routines, for-loops, while loops, variables, and possibly other things
- Let's write a shell script that determines the factorial of a given argument (> 0).
- Let's start by writing the main body of the program

```
#!/bin/bash
i=$((${1}-1))
total=$1
while [ "${i}" -gt 1 ]; do
   total=$((${total} * ${i}))
   i=$((${i}-1))
done
echo "${1} factorial is ${total}"
```

- But what if `${1}` isn't > 0? We should check this.

```
#!/bin/bash

if [ "${1}" -lt 1 ] # line *
then
  echo "${1} is not > 0" 1>&2
  exit 1
fi
# Insert body of function here
```

- Why do we encapsulate `${1}` in quotes on line *?
- What else should we check?
  - That we have a single parameter? Yes.
  - That the single parameter is a number? Yes. But that's more complicated so we won't

```
#!/bin/bash
if [ ${#} -ne 1 ]
then
  echo "Incorrect number of parameters" 1>&2
  echo "Usage: ${0} n, where n > 0"
  exit 1
elif [ "${1}" -lt 1 ]
then
  echo "${1} is not > 0" 1>&2
  echo "Usage: ${0} n, where n > 0"
  exit 1
fi
# Include body here
```

- But now we're duplicating lines of code. What's the solution ? Create a usage routine and call that!

3 Testing

- We're going to perform a miniature case study of testing and determine some possible test cases for a problem.
- Some things to keep in mind:
  - In this course, when we ask you to test you should be less concerned with invalid input (e.g. if we tell you that all input should be greater than 5, input less than 5 is invalid) unless we tell you otherwise

  - Testing is *hard*. If you get stuck on a Marmoset test case, don't waste hours on it. Move on, it's only worth a very small fraction of your final mark
  - Try to make sure your tests have good coverage. Ask yourself if you've looked at boundary/edge cases, corner cases, equivalence classes (e.g. different tax brackets), weird cases (error guessing - what assumptions could a person make to make their solution wrong)?
  - Sometimes the "No one could make this mistake" type test cases, are the ones that are needed the most (They work as sanity checks).

- **Problem:** Given a program that reads from stdin a list of integers with the goal of determining if some combination of a list of integers can sum to the last integer given, where integers are >= 0, e.g. input is of the form

```
n
x_1
x_2
..
x_n
y
```

  where n specifies the number of possible summands, `x_i` is a possible summand in ascending order, and y is the target value. If no combination of integers can sum to the target value then "Impossible!" should be printed.

- What are some possible test cases?
  - Equivalence classes
    - small, medium, and large values of the target (Problem: Define small, medium, and large.)
      - Choose values that seem seem reasonable, you don't need to try 4 billion possible values
  - Boundary/edge cases
    - Test containing 0 as target and no 0 summand should print "Impossible!"
    - Test containing 0 as both target and summand should work
  - Corner cases
    - Test which contains even integers and an odd target should print "Impossible!"
    - Target smaller than all integers should print "Impossible!"
  - Weird cases/error guessing:
    - Test which fails if you start from low and go to high
    - Target less than largest integer but target still attainable
  - Sanity check:
    - Targest is in list

- What are some "bad" test cases?
  - List of integers is in descending order or no order
  - Any value is negative
  - More than n summands
  - Less than n summands

4 Compiling Your First (C++) Program

- Let us suppose we have the following C++ program (`avg.cc`):

```cpp
#include <iostream>

using namespace std;

int main(int argc, char* argv[]){
  double val = 0.0, avg = 0.0;
  int num_vals = 0;
  while(cin >> val){
    avg += val;
    ++num_vals;
  }
  avg /= num_vals;
  cout << "Average value: " << avg << endl;
  return 0;
}
```

- What does it do? It computes the average of a list of numbers provided on standard input.
- How do we compile it? `g++ avg.cc`
- And then we can run it with some input!

```
> ./a.out
5
4
3
2
1
Average value: 3
```

- What if we want to give the executable a different name? Use the `-o` option.
  - `g++ avg.cc -o avg`