Goals

- Learn a little more C
- Write a simple MIPS programs
- Compute a benchmark score for an imaginary processor
- Practice math and conversions with numbers in varying bases

For parts 1 and 2, paste your programs and program output (including the gcc and execute commands that you used to compile and run it) into a document (PDF or open office or plain text). Upload this document to Sakai and hand in a hard copy by the due date. BE SURE YOU PUT YOUR NAME AND THE LAB NUMBER at the top of the document!

Also upload the programs themselves as separate files.

Please leave the “Display name” unchanged when using the Sakai “upload file” feature.

Part 1: Some More C [30 points, 15 each]

Chapter 4 of K & R discusses functions in C. Chapter 5 deals with arrays and pointers. You should probably read these before doing the next two problems.

All C programs must be FULLY COMMENTED—header comments should provide information about author, date, program purpose (input/output and summary of processing performed); in-code comments should describe variables and summarize major steps in the algorithm. Indent properly (imitate K&R). Use white space, e.g., blank lines, to make the program more readable.
1. Write a C program that implements the “Guess My Number” game. The program should choose a number between 1 and 100 inclusive, and prompt the user to guess that number. After each guess, the system should print whether the guess was too high, too low, or correct. If correct, the program should print the number of guesses that the user made, and then prompt the user to play again.

Sample run:

```
aldenv27:lab2 jwenskovitch$ gcc lab2part1.c -o lab2part1
aldenv27:lab2 jwenskovitch$ ./lab2part1
Your Name
Lab 2
Tue Sep 9 08:19:42 2014

Hello user! I’m thinking of a number between 1 and 100.
See if you can guess it!
Your guess: 50
Your guess was too high!
Your guess: 45
Your guess was too high!
Your guess: 40
Your guess was too low!
Your guess: 42
You got it in 4 tries! My number was 42.

Would you like to play again (y/n)? y

Hello user! I’m thinking of a number between 1 and 100.
See if you can guess it!
Your guess: 19
Your guess was too high!
Your guess: 7
You got it in 2 tries! My number was 7.

Would you like to play again (y/n)? n
```
2. Write a C program that has a `main` function and a function named `rotate`. The `rotate` function should have three parameters: an `int` array, an `int` containing the size of the array, and an `int` amount. The function should rotate the elements in the array by the stated amount. If the amount is positive, the array should be rotated to the right; if the amount is negative, the array should be rotated to the left. For instance, if the array is named `x` and it contains \[ 2 \ 3 \ 5 \ 7 \ 11 \] and we call `rotate(x, 5, 2)`, the resulting array `x` will look like \[ 7 \ 11 \ 2 \ 3 \ 5 \].

Your function should be “smart”—it should recognize that rotating by a multiple of the size of the array is equivalent to doing nothing at all (so, in the previous example, “`rotate(x, 5, 25)`” should not do any work at all) and that rotating by an amount greater than the array size (in absolute value) is equivalent to rotating by something that is smaller in absolute value. For instance, “`rotate(x, 5, -19)`” is equivalent to “`rotate(x, 5, -4)`”. There are even more ways to make the rotation faster, but these are the only ones you are required to consider.

There is a way to do this by creating a temporary array; however, you have not yet seen how to create arrays of variable size, so I request that you NOT use this approach! Your solution should rearrange the elements of the given parameter array without using a second array. Efficiency is not a concern (apart from the comments in the previous paragraph).

In `main`, create three `int` arrays of sizes 10, 15, and 25. Initialize them with the positive integers 1, 2, ... . Call `rotate` three times for each array (a total of nine times), using rotation amounts 4, -30, and 17 for each array. Reset each array to its original values prior to each call! Print each array after returning from the function, with appropriate labelling.

You may find it useful to have two functions for printing the arrays and for resetting the arrays.

Sample run:

```
aldenv27:lab2 jwenskovitch$ gcc lab2part2.c -o lab2part2
aldenv27:lab2 jwenskovitch$ ./lab2part2
your name
Lab 2
Tue Sep 9 09:07:27 2014

After rotating by 4, array a =
7 8 9 10 1 2 3 4 5 6
After rotating by -30, array a =
1 2 3 4 5 6 7 8 9 10
... etc. ...
After rotating by 17, array c =
9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8
```
Part 2: MIPS [10 points]

3. Write a fully-commented MIPS program that shows the calculation:

\[ x = a - (-b - c + d) - 3 \]

where \( a \), \( b \), \( c \), and \( d \) have the values 10, 5, -8, and 6, respectively.

Each of the variable values should initially be stored in the `.data` section of your code and loaded into registers as needed. Use the `t` registers to hold your temporary calculations. The final result of the calculation should be printed to the console.

Imitate the example on pages 67–68 of the HP book.

The last two lines in your MIPS program should be:

```
li $v0,10
syscall
```

These properly terminate the program.
Part 3: Performance [10 points]

4. Below is an incomplete table showing a partial set of benchmarks for an imaginary processor. First, complete the table by calculating both \( \text{ExecTime} \) and \( \text{SPECratio} \) for each component program in the benchmark.

Use the following formulas:
\[
\text{ExecTime} = IC \times CPI \times \text{CycleTime}
\]
\[
\text{SPECratio} = \frac{\text{Ref Time}}{\text{Exec Time}}
\]

Next, calculate an overall \( \text{SPEC} \) score for this processor by taking the geometric mean of the five \( \text{SPECratio} \) scores that you calculated. To calculate a geometric mean, multiply all five \( \text{SPECratio} \) scores together, then take the 5th root of that result.

\[
\text{SPEC} = \sqrt[5]{\text{SPECratio}_1 \times \text{SPECratio}_2 \times \text{SPECratio}_3 \times \text{SPECratio}_4 \times \text{SPECratio}_5}
\]

Congratulations, you have just calculated your first benchmark!

<table>
<thead>
<tr>
<th>Process</th>
<th>IC ( \times 10^9 )</th>
<th>CPI</th>
<th>Cycle Time (ns)</th>
<th>Exec Time</th>
<th>Ref Time</th>
<th>SPECratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>perl</td>
<td>2118</td>
<td>0.84</td>
<td>0.40</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bzip2</td>
<td>2389</td>
<td>0.18</td>
<td>0.40</td>
<td>1250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>1050</td>
<td>1.44</td>
<td>0.40</td>
<td>1800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td>336</td>
<td>1</td>
<td>0.40</td>
<td>750</td>
<td></td>
<td></td>
</tr>
<tr>
<td>go</td>
<td>1658</td>
<td>0.77</td>
<td>0.40</td>
<td>1500</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part 4: Number Conversions [10 points, 1 each]

Perform the following addition or subtraction, or convert the following numbers from the source base to the target base. Please show enough of the internal steps in your computation to prove that you didn’t just use a calculator to convert the numbers.

5. \((100101)_2 + (10111)_2\
6. \((9AF)_{16} + (332)_{16}\)
7. \((123)_8 + (765)_8\
8. \((742)_{16} - (643)_{16}\)
9. \((1000)_{10} = (x)_2\)
10. \((1000)_{10} = (x)_{16}\)
11. \((FEDC)_{16} = (x)_{10}\)
12. \((100100101010010)_{2} = (x)_{10}\)
13. \((87AB71)_{16} = (x)_2\)
14. \((5632)_{16} + (110010101010)_{2}\)