

CISC181H Spring 2009 Lab01

- Write a program for each of the following problems. You may work in groups of two or three. Be sure to save every separate program. All programs must be properly commented and indented (see Assignment Standards on the class website). Ask your TA for guidance.
- Think about what you want to do, and then look up how to do it in Savitch. For some problems I included the C math library in my code so I could use math functions from there: `#include <cmath>`
- Name each program lab01.n.c, where n is the number in the list below. For example, the name of the file for the first will be lab01.1.c. Put the files in your lab01 directory (see Anderson for instructions on how to make a directory).
- Do you remember how to edit, compile, execute, and script a C++ program? If not, see lab00, but do not submit it or turn it in.
- Design fully, then code and test incrementally! In other words, do not type a whole program into the machine, ever. Put in little bits (1-3 lines), then compile and test. This usually works best if you start with the inside (middle) of the program. Save any I/O for last.

Programs

1. Write a program that uses C++ iteration structures to produce an approximation of a diagonal line. See the attached example of output. Your output does not have to be identical, but must look at least this nice. Your program should use all three kinds of iteration structures in C++, and should use each in a logical way¹.
2. Use iteration to print out major scales, using the cycle of fifths. There are twelve possible notes, and major scales skip notes in a certain pattern (1,1,0,1,1,1,0). The twelve notes are
C, C#, D, D#, E, F, F#, G, G#, A, A#, B
and using the skipping pattern, an eight-note major scale starting on C would include
C, D, E, F, G, A, B, C. A major scale starting on G would include *G, A, B, C, D, E, F#, G*.
Print twelve major scales, each seven steps up from the previous², so that after C would be a scale starting with G, then D, etc.
Design fully, but then code incrementally, testing as you go. Remember, hours of coding can save minutes of planning!
3. On Strauss, automatically allocate space for 3 chars, 3 ints, and two doubles. Then print out the address of each variable³. Use graph paper or other constant scale medium to create a picture of the space these variables use on the stack, showing byte addresses. Show the stack as four bytes wide and label each variable with the address shown in the script of your program running. Recall that the addresses may change each time you run the program, so don't finish your drawing until you have scripted the execution.

¹Do not use "break" or "continue" in loops in this course. These are crutches - any loop written with them can also be written without them, which keeps the control of the loop at the top or bottom where it is easy to find.

²So why is it called the cycle of fifths?

³Remember that we had to use a trick to see the address of the char address, instead of the contents, when we used "<<".

Now dynamically allocate space for an int. Approximately how far, in bytes, is the dynamically allocated memory (heap) from the automatically allocated memory (stack)?

Now allocate space by creating a char array that holds a string literal (those double-quoted things). Is this space on the stack, the heap, or somewhere else?

4. (Adapted from Tan & D’Orazio, C Programming) Consider building an airport with the runway built on landfill. The contractor has two dump trucks, one with a capacity of 8 tons and the other with a capacity of 12 tons. The contractor uses the trucks to haul fill from a remote site to the airport location. The operating cost per trip for the 8 and 12 ton trucks is \$14.57 and \$16.26, respectively. One truck cannot make more than 60 percent of the total trips.

Share total cost figures with each other as you approach a solution. Try these ton values: 2,10, 20, 30, 65, 72, 80, 81, 83, 84, 101, 115, 121, 1000, 1051 to test your program. Write a test function that includes these tests and calls the function you will write to solve the problem.

Write a program that iterates to develop the minimum cost for a given number of tons. Prompt the user to enter the total number of tons. Display the number of trips required for each truck and the total cost.

Comments on the problem:

The problem definition isn’t clear on what to do with less than 8 tons, or between 8 and 12 tons. Common sense dictates that for loads of less than or equal to 12 tons, use only 1 truck (the least cost truck). Another reasonable exception to the 60/40 rule is when the number of trips between the two trucks differs only by 1. So, for example, for small loads, 2 trips with truck A, and 1 trip with truck B is permissible, even though that is not less than or equal to a 60/40 split. Note also that the 60/40 split is expressed in the number of trips, not in tons.

You should have a total of 4 programs named lab01.1.c to lab01.4.c. Make a single script file (see lab00 for the instructions) where you cat, compile, and run each one in its final form.

Submit all 4 program files *and* your script on WebCT by midnight before your next lab. Give the paper version of the complete script file **only** on paper to your TA at the **beginning** of your next lab. Note: Cat, compile, and run each program in order - do *not* cat all programs, then compile, etc.

```
terry% g++ -o diagonals diag.cc
terry% diagonals
Enter height and width: 3 3
*
*
*
Enter zero to quit, or 1 to continue: 1
Enter height and width: 10 3
*
*
*
*
*
*
*
```

```
*
*
*
Enter zero to quit, or 1 to continue: 1
Enter height and width: 4 6
**
**
*
*
Enter zero to quit, or 1 to continue: 1
Enter height and width: 3 10
****
***
***
Enter zero to quit, or 1 to continue: 1
Enter height and width: 3 11
****
****
***
Enter zero to quit, or 1 to continue: 1
Enter height and width: 4 1
*
*
*
*
Enter zero to quit, or 1 to continue: 0

terry%
```