CS900 Lab - Day 04
July 3st, 2003
Instructor: Thomas Hofmann

New Topics Covered

Loops – Lecture: Day 4, King: Chapter 6
Arrays, Lecture: Day 4, King: Chapter 8

Installing

Copy the template source files from the directory /course/cs900/day04

Problem 1 – Squares

Write a program that prompts the user for an integer n, and then prints out all even squares greater than 1 and less than or equal to n. A sample run of the program might look like this (user input is in italics).

```
Please enter an integer.
> 100

Squares are:

     2^2 = 4
     4^2 = 16
     6^2 = 36
     8^2 = 64
     10^2 = 100
```

Implement your program in your squares.c file. To compile, type gcc -Wall -o squares squares.c, and to run, type squares. Remember to fill in the header and to properly comment your program.

Problem 2 - Reverse

Write a program that takes in an arbitrary-length number (1 or more digits) and reverses it. You may not ask the user for length of the number; you must simply take in an integer and spit it back out with its digits reversed.

Hint: Think about a kind of loop that will allow you to DO something, and then continue to do it WHILE a certain condition is true.

Implement your program in your reverse.c file. To compile, type gcc -Wall -o reverse reverse.c, and type reverse to run. As always, remember to fill in the header and to properly comment your code.

Problem 3 – Numbers

Write a program that reads in an arbitrary-length series of digits separated by white space and then prints out the digits as one, complete number. The user will signify the end of the string of digits by entering a negative number. One run of the program might look like this:

```
Enter your digits: 5 6 2 9 -1
The number you have entered is 5629
```

NOTE: Within your program, you must turn the single digits into a single integer. You cannot just print out the digits with no space between them.

Warning: This program is tougher than it looks. Implement your program in the num.c file. Compile using gcc -Wall -o num num.c, and run by typing num.

Problem 4 – Perfect Numbers

Write a program to find all perfect numbers up to some user-specified limit.

A perfect number is a number which equals the sum of its proper divisors (also called *aliquot parts*). A proper divisor that divides the number evenly, but is not the number itself.

In order to accomplish this task, write a function that computes the sum of all proper divisors of a given number. This function can be called in a for loop cycling from 1 to the specified upper limit.

Hint: The first four perfect numbers should be:

```
6    = 1 + 2 + 3,
28   = 1 + 2 + 4 + 7 + 14,
496  = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248
8128 = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 127 + 254 + 508 + 1016 + 2032 + 4064
```

The next ones are:

```
33550336
8589869056
137438691328
2305843008139952128
2658455991569831744654692615953842176
191561942608236107294793378084303638130997321548169216
```

How far can you get? Implement your program in a file perfect.c.

Problem 5 – Random Permutations

Write a program that will read an array of *M* numbers - *M* being a pre-defined constant – and outputs 5 random permutations (shufflings) of these numbers. (Which means for each shuffling you have to re-arrange the numbers in some random order.)

Hint: You should use the `rand()` *function of the Standard C Library to implement* `randNumber(int)`*. (Look at Appendix D in your book. It details the Standard Library.) Remember to include stdlib.h so you have access to the Standard Library functions.*

```
#include <stdlib.h>
```

An example run might look like this:

```
Input a sequence of 10 numbers separated by blanks
> 0 1 2 3 4 5 6 7 8 9
Random permutation #1:
5 3 0 4 2 6 9 1 7 8
Random permutation #2:
0 1 5 3 4 7 9 2 6 8
Random permutation #3:
6 8 5 7 4 3 0 2 1 9
Random permutation #4:
6 0 8 9 2 3 7 1 5 4
Random permutation #5:
9 6 8 2 1 0 7 4 3 5
```

Implement your functions in your perm.c file. In your `main()` function, write some code that will read a sequence of 10 integers and outputs a random permutation.