CS900 Lab - Day 03
July 2st, 2003
Instructor: Thomas Hofmann

New Topics Covered

Arithmetic expressions – Lecture: Day 2, King: Chapter 4
Logical expressions and conditional statements – Lecture: Day 3, King: Chapter 5

Installing

Copy the following files from the directory /course/cs900/day03: isbn.c, calculator.c and adder.c. You can do the following:

- Login using your login name and password
- Create a subdirectory day03 in your home directory by typing in shell:

```
cd
mkdir day03
cd day03
```

- Copy over the files

```
cp /course/cs900/day03/*.c .
```

Problem 1 – ISBN Numbers

In your isbn.c file, write a program that checks the validity of ISBN numbers. Every published book has an ISBN number, normally right above the barcode. Take a look at the back of your textbook. The ISBN number is 0-393-96945-2. To avoid bad scans of ISBN numbers, the 10 digits have the following property:

$$(10d_1 + 9d_2 + 8d_3 + ... + 2d_9 + d_{10}) \bmod 11 = 0$$

Where d_1 through d_{10} are the 10 digits of the ISBN. (In reality, this is a bit more complicated, since the last digit may also be a 'X", but we simplify this here for our purposes.) Your program should take in a 10- digit ISBN and determine whether or not the number is valid. Two sample runs are shown below (user input is in *italics*):

Example 1:

```
Please enter an ISBN number (no dashes or spaces):
> 0393969452
VALID
```

Example 2:

```
Please enter an ISBN number (no dashes or spaces):
> 0393969451
INVALID
```

Hint: You can use a generalization of the following placeholder semantics for scanf to read in a single digit (into an int variable dig):

```
scanf("%1d",&dig);
```

Compile using `gcc -Wall -o isbn isbn.c`. and run by typing `isbn`. Remember to fill in the header and to properly comment your program.

Extension

Extend the program in the following way. The user enters a 10 digit number and specifies a position where the digit has been corrupted. Correct the corrupted digit so that the resulting number is a valid ISBN number and output the number.

Problem 2 - Simple Calculator

Write a program that simulates a simple calculator that functions as follows: You will prompt the user to enter 2 numbers, one at a time. You will then ask the user to enter a number corresponding to an operation, with a menu of possible inputs. Make sure you are able to accommodate non-integral decimal operands and invalid user input. Here are 2 sample runs of the program:

Example 1:

```
Enter first number: 4.2
Enter second number: 5

Enter a   1 for +
          2 for -
          3 for *
          4 for /

Entry: 1

The answer is 9.2.
```

Example 2:

```
Enter first number: 1
Enter second number: 3

Enter a   1 for +
          2 for -
          3 for *
          4 for /

Entry: 5

Invalid operator. Exiting.
```

Hint: You could use if/else statements to do the decision making for this program. However, you will have a cleaner program if you use a switch statement.

Implement your program in the calc.c file. To compile, use `gcc -Wall -o calc calc.c`, and to run, type `calc`. Be sure to fill in the header and to properly comment your program.

Extension

Implement additional functionality that you consider useful.

Problem 3 – Full ADD ahead!

In class we have discussed the implementation of half and full bit adders using logical circuits. Here we would like to simulate such circuits using C. We want you to write a program that consists of two functions and a user interaction part to read in the arguments. The two functions will implement a full adder by computing the sum bit and the carry bit. The integers encode Boolean values (false and true) as numbers (0 and 1)

```
int fullAdderSum   (int A, int B, int C);
int fullAdderCarry (int A, int B, int C);
```

Finally read two 4 bit numbers from the keyboard and compute their sum using the implemented functions. A user interaction might look like this

```
First number:  1001
Second number: 1011
Carry bits:   1011-
Result    :   10100
```

Implement your program in a file adder.c.

Extension

Read the lecture slides about the NAND function. NAND gates can be used to implement AND, OR, and NOT gates. Implement a function NAND in C

```
int nand (int A, int B);
```

and use this function to implement functions `fullAdderSum` and `fullAdderCarry` without using any additional Boolean operators.