CS270 Programming Assignment 1
“Radix Conversions”

Due Friday, September 9 (via checkin by 3:00pm)
Homework and programming assignments are to be done individually.

Goals
In this assignment, you will learn C programming and reinforce your understanding of number representation by implementing C functions to convert numbers from any specified radix into decimal, to convert decimal numbers into any specified radix, and ultimately, to convert a number in one specified radix into another specified radix. A “specified radix” in this program can range from 2 to 36.

The Assignment
Make a subdirectory called PA1 for the programming assignment; all files must reside in this subdirectory. We have provided you with a number of files to complete the assignment. Copy the following files into your PA1 directory:

```
http://www.cs.colostate.edu/~cs270/Assignments/PA1/main.c
http://www.cs.colostate.edu/~cs270/Assignments/PA1/myfunctions.h
http://www.cs.colostate.edu/~cs270/Assignments/PA1/myfunctions.c
http://www.cs.colostate.edu/~cs270/Assignments/PA1/Makefile
```

You will need to implement the following functions inside “myfunctions.c”. The skeleton structure has already been provided for you.

```
1) int radixNToDecimal(char radixNNumber[], int radixN);
2) char* decimalToRadixN(int decimalNumber, int radixN, char result[]);
3) char* radixAToRadixB(char radixANumber[], int radixA, int radixB, char result[]);
```

Function 1 – radixNToDecimal:
Use Horner's Algorithm (as reviewed in lecture 08/25/11) to convert a null-terminated array of char (i.e., a C string) into a decimal number (radix = 10). The array of char, radixNNumber, represents a non-negative number in the base indicated by argument radixN—convert this number into radix 10 and return it as an integer.
To complete this function, you will need to utilize a provided helper function, symbolToDecimal, which converts a digit from radixN into its decimal equivalent, e.g., 0 → 0, 1 → 1, …, 9 → 9, A → 10, B → 11, …, Z → 35. This provides 36 unique symbols which we will use to represent numbers with bases ranging from 2 to 36.

Function 2 - decimalToRadixN:
Use repeated division and modulo operations (as reviewed in lecture 08/25/11) to convert an integer into a null-terminated array of char (i.e., a C string). The integer, decimalNumber, represents a non-negative number in radix 10, and must be converted into the radix specified by argument radixN and returned as an array of char.
To complete this function, you will need to utilize another provided helper function, decimalToSymbol, which converts a decimal value into its radixN equivalent, e.g., 0 → 0, 1 → 1, …, 9 → 9, 10 → A, 11 → B, …, 35 → Z.
Finally, this function will require you to build and return a null-terminated string (i.e., an array of char). The char array result[] has been provided for you (as a function argument) to accomplish this. The size of the array result[] is PLACES_LIMIT.

Function #3:

At last, use functional composition to implement this final function in terms of the previous two functions. This will provide a function which converts a non-negative null-terminated array of char representing a number in radixA into an equivalent non-negative null-terminated array of char in radixB. Again, the char array result[] has been provided (as a function argument) to help you accomplish this.

Compile and Run:

We provided you with a Makefile to compile the program. The program is executed as follows, and takes three arguments:

```
%> pa1 <number> <fromRadix> <toRadix>
```

Where *number* is the number to convert, *fromRadix* is the radix *number* is currently represented in, and *toRadix* is the radix to convert *number* into.

Try the program with the following test cases:

```
 %> pa1 255 10 2
 %> pa1 10101100 2 16
 %> pa1 RADIX 36 10
```

Calculate these results by hand and verify if you are getting the correct output. The results should be “11111111”, “AC”, and “45833721”, respectively. We will run your program using different inputs, so DO NOT HARDCODE values!

For this assignment you must also submit a README file with your name and answers to the following questions. Copy the question into the file and then type in the answer after the question.

Question 1: Are you doing your assignments on the school machines or at home? If at school what is the name of the machine you are using to answer these questions?

Question 2: Type gcc --version on the command line and write down the output. This assumes Linux, if you are running on another operating system, then write down the compiler version.

Submission Instructions

When you are done, your directory should have main.c, myfunctions.c, myfunctions.h, Makefile and a README file. To package the files into a single compressed file, type the following command from inside PA1 directory:

```
%> cd PA1
%> make pack
```

This will create a file called PA1.tar.gz one directory above PA1. All assignments will be submitted directly via checkin, which will be explained and demonstrated in recitation. A sanity check of your PA1.tar.gz will ensure that your submission has all the required files:

```
%> mkdir ~/Temp
%> cp PA1.tar.gz ~/Temp
%> cd ~/Temp
%> tar -zxvf PA1.tar.gz
%> ls PA1
```

Grading Criteria
Points will be awarded as follows: functionality - 75 points (30 for function #1, 30 for function #2, and 15 for function #3), coding style and comments - 10 points, following assignment directions - 5 points, and supplying answers to the README questions - 10 points. The grading factors we consider for coding style include having clear and concise comments, consistent indentation, and the minimal amount of code to solve the problem. You will also need to ensure that every function (declared in myfunctions.h) has a properly formatted description header.

Late Policy
Late assignments will be accepted up to 48 hours past the due date with a deduction of 10% per 24 hours. Late assignments will not be accepted after 48 hours. Please contact the instructor or teaching assistant if you have problems with checkin.