CSc 352 Summer03 Assignment 3

Start: 06/23/03
Due: 06/30/03 (9:00pm)
Turnin ID: cs352_assg3

In this assignment, you are going to turn in the following files using **turnin** command:

`numio.c numcalc.c numerr.c calculator.c det.c hanoi.c`

0. Background readings:

Please read the information about the following C standard functions from the textbook, man page, online documents on the class website or other online materials searched on google.

- fgets()
- getchar()
- scanf()
- printf()
- exit()

1. High resolution calculation:

This problem is an extension of problem 2 in assignment 2.

In this problem, you are to implement a C program that can do +, -, *, / and ! operations on huge integers you stored in the previous assignment.

(1) First, you need to modify your code (**numio.c**) from the previous assignment to enhance the error handling capability. The possible error codes are defined in the header file (**numio.h**).

You should return the corresponding error code whenever an error is encountered. This time, in functions like **read_num()** and **print_num()**, you should **NOT** assume the input from the user or the parameters passed into the functions are all legal. (i.e. the **num** parameter passed to **print_num()** may be a NULL pointer, which is regarded as an error.) Also the input from the **stdin** may be longer than **MAX_DIGITS** digits, and that should be handled and shouldn't cause memory violation. You may need to use some other function other than scanf() to get the input from **stdin**.

You also need to delete the **main()** function in file **numio.c** you implemented in the previous assignment.

(2) To facilitate the error handling, you need to write a function:

```
void print_errmsg(int ret, char *msg);
```

This prototype is in file **numio.h**. The first parameter **ret** for this function is an status code returned from **read_num()**, **print_num()**, or any new functions we are going to implement in this assignment. The second parameter **msg** is a user specified message string. **print_errmsg()** will print out the application error message according to the parameter **ret** followed by the user's message string **msg** into the **stderr!** The application error message and the user message are concatenated by a character "**:**". For example, if the application error message is "OK" (corresponds to error code **CERR_OK**) and the user message **msg** is "I am working fine.", your function call is **`print_errmsg(ret, "I am working fine")`**, where **ret** has the value **CERR_OK**. Then you need to print out:

```
OK:I am working fine.
```

We will have a table of all the error code and their corresponding application error messages at the end of the specification of this problem. You need to use the exact error message listed in the table. Otherwise, your output will be different from expected. You can make sure all your output is correct by comparing with the output produced by the provided executable.

This **print_errmsg()** function is implemented in a file **numerr.c**.

(3) Write a function for each of the 5 operations +, -, *, / and !. Prototypes:

```
int op_plus(unsigned char *num1, unsigned char *num2,
            unsigned char *result);
int op_minus(unsigned char *num1, unsigned char *num2,
            unsigned char *result);
int op_mult(unsigned char *num1, unsigned char *num2,
            unsigned char *result);
int op_div(unsigned char *num1, unsigned char *num2,
            unsigned char *result);
int op_fact(unsigned char *num1, unsigned char *result);
```

All these prototypes are provided in a given header file **numcalc.h**. You are going to implement all these function in a file **numcalc.c**.

For all the operations, if the input huge integers **num1** or **num2** contain errors identified by**, CERR_DIG_ILLEGAL or CERR_LEN_ILLEGAL,** return the corresponding error code without doing anything. If the parameter **result** is a NULL pointer, return **CERR_NUM_NULL** without doing anything. The result of the operation should be filled in the char array pointed by **result**.

fact operation here means the factorial operation. n! = n*(n-1)*(n-2)*(n-3)*…*2*1

For **plus, fact** and **mult** operations, if the result contains more than **MAX_DIGITS** digits, return error code **CERR_OUTOF_BOUND**.

For **minus** operation, if the **num1** is less than **num2**, return error code **CERR_OUTOF_BOUND**, because we don't handle negative huge integers in this assignment.

For **div** operation, we discard the reminder part in dividing operation. For example: for a calculation "10 / 3", the result should just be "3" and the reminder 1 is discarded.

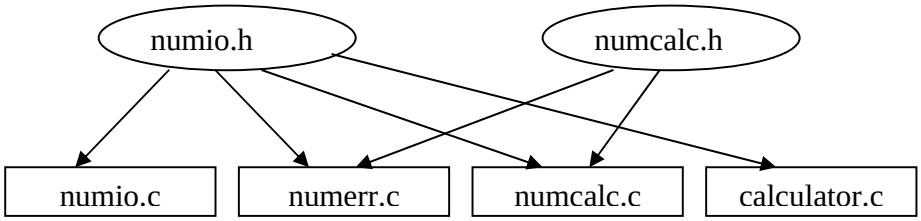
(4) Write the main User Interface (**UI**) part of the high-resolution calculator in file **calculator.c** now. Your **main()** function will be in this file. The UI is just an infinite loop: read the operator from the **stdin**, read the operands for the operator (for "!" operation, there is only one operand) from the **stdin**, call the specific **op_xxx()** function to do the calculation and print out the result or error message. Repeat this sequence of actions until the user specify "@" as the operator. "@" as an operator will make the program terminate by returning 0 in the **main()** function. You may assume all the operators you got from **stdin** are one of +, -, *, /, ! or @.

In this infinite loop, whenever a function call returns status (for example, we store the return value in an **int** variable **ret**) not equal to **CERR_OK**, print the error message by calling **print_errmsg(ret, "my msg")**, then execute statement "**continue;**" to start over instead of exiting the whole program. (You may notice, here we fixed the user message for the second parameter of **print_errmsg()** function to be constant "my msg". I just design the API like this to show in many real world C projects, the error message facility usually provide a way of adding the users' own message in the diagnostic output. We fix it here as "my msg" just for the convenience of grading.)

(5) Finally, arrange all your files for this project in the same directory and compile then into one program. You have the files showed in the table below. And for each file, I list the functions you have to implement in that file. Besides the required functions in each file, you can also define and implement your local helper functions, which should be defined as **static**.

Filename	Contents in the file
numio.h	Constants and function prototypes for integer input and output related use.
numio.c	Definition of integer input and output related functions: read_num(), print_num().
numerr.c	Definition of error handling functions: print_errmsg().
numcalc.h	Constants and function prototypes for integer calculation related use.
numcalc.c	Definition of integer calculation related functions: op_plus(), op_minus(), op_mult(), op_div() and op_fact().
calculator.c	UI part, including main() function.

The header file dependence is like this:



Then compile and link the program by either of the following methods:

Method 1:
gcc –c numio.c numerr.c numcalc.c calculator.c –I/home/cs352/summer03/assignments/hw3
gcc –o calculator numio.o numerr.o numcalc.o calculator.o

Method 2:
gcc –o calculator numio.c numerr.c numcalc.c calculator.c –I/home/cs352/summer03/assignments/hw3

Then you will get an executable "calculator".
A sample run of this program is as the following:

```
% calculator
+ 11111 22222
=33333
* 22a 3333
Illegal digit:my msg
! 10
=3628800
@
%
```

(6) Error message table:

Error Code	**Message to be printed by print_errmsg()**	**Description about the error**
CERR_OK	"OK"	The function is executed correctly.
CERR_NUM_NULL	"Null pointer"	The variable representing a char array is a NULL pointer.
CERR_DIG_ILLEGAL	"Illegal digit"	There is an illegal digit (some number that is less than 0 or greater than 9 or a non-digit character) in the char array representing the huge integer.
CERR_LEN_ILLEGAL	"Illegal number length"	The huge integer has illegal number of digits (either less than one or greater than **MAX_DIGITS**).
CERR_OUTOF_BOUND	"Number out of boundary"	Either the calculation result has more than **MAX_DIGITS** digits, or an operation is going to produce a negative result.
Any other value	"Unknown error"	

2. Determinant of the square matrix:

In this problem, you are going to write a C program in a single file **det.c**, which calculate the determinant of a given square matrix (which has the same number of rows and columns). Please refer to the following webpage for a description about matrix and matrix determinant:
http://easyweb.easynet.co.uk/~mrmeanie/matrix/matrices.htm (**PS:** please notice that in the Determinant section of this webpage, the first example of calculating the determinant of a 2*2 matrix is wrong. It should be $\det\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - cb$)

The input is from the **stdin** and is in the following format:

```
2
3 2
5 6
```

The leading "2" means this matrix is of size 2 by 2 (2 rows and 2 columns). Then the following is the matrix. There is a space between every 2 elements. If he size is not in [1, 50] or if the following matrix is not correct (there are missing or extra rows or columns), print "Wrong input\n" into **stderr** and "**exit(1)**". You may assume all the inputs are integers and there won't be letters, strings or other symbols.

Otherwise, calculate the determinant of the given square matrix and print it out. In the above example, the output should be like:

```
8
```

After printing the result, "**return 0;**" from **main()**.

3. Hanoi Tower:

Please refer to the link http://www.cut-the-knot.org/recurrence/hanoi.shtml for a description of the "Hanoi Tower" game. You are going to write a C program **hanoi.c**. It takes a single integer **n** as input from the **stdin**. **n** should be in [1, 64], otherwise print "Out of boundary\n" into **stderr** and "**exit(1);**". You may assume the input is always an integer and there won't be letters, strings or other symbols.

If the input **n** is correct, we begin the game. Assume there are **n plates** on **pole A** and you are going to print the steps to move all those plates to **pole C** with the aid of **pole B**. For example: n=3, you are going to print:

```
A C
A B
C B
A C
B A
B C
A C
```

In each row, the first letter is the source and the second letter is the target. For example "A C" means "moving the top plate on **pole A** to **pole C**".

When you test or debug your program, please use small values for n at the very beginning. Otherwise you will be occupying a large portion of the processor on lectura, since this program is quite CPU intensive. If that happens, people working on lectura will be beating you ☺

About compilation:

Read the information about gcc command line options about preprocessor control at http://gcc.gnu.org/onlinedocs/gcc-3.3/gcc/Preprocessor-Options.html#Preprocessor%20Options

You don't have to copy the provided header files to your own directory. In stead, please do the following things:

- Use #include <**numio.h**> and #include <**numcalc.h**>, instead of #include "**numio.h"** and #include "**numcalc.h**".
- When compile with gcc, specify ***–Ipathname*** option, where *pathname* point to the hw3 directory in the cs352 home where the provided headers are stored. In this assignment *pathname* is /home/cs352/summer03/assignments/hw3. With this ***–Ipathname*** option, the *pathname* is added to the header file search list.

An advantage of doing this is that you will always use the most updated header files. You don't have to always copy header files to your own directory. It's also easy for us to change constants in the header file when grading. So:

- Always use the constants defined in the provided headers by including the header files.
- Do NOT use magic numbers in the program if there is a correspondent macro constant defined in the provided header files.
- For those macro constants we defined in the provided header files, do NOT redefine them in your own files.