EECS 10: Assignment 3

Prepared by: Che-Wei Chang, Prof. Rainer Dömer

July 14, 2014

Due Monday July 14, 2014 11:00pm

1 Guess the Number [20 points]

Write a program that plays the game of "guess the number". In this game, the computer "thinks" of a random number between 0 and a user-specified upper limit and the player has to guess this number. The computer will help the player by giving hints on whether the guessed number is less than or greater than the chosen number.

At the beginning, the computer asks the player for the upper bound for generating the random number. Your first line of output should display:

Enter the upper bound for the random number:

Once the user has entered the upper bound (say n=1000), your program chooses the number to be guessed by randomly selecting an integer in the range of 1 to n. The program then displays the following:

*"******Guessing Game******"*
I have selected a number in the range of 1 to 1000.
Can you guess the selected number?
Try No.1, please input the number:

The player then types a first guess. The program responds with one of the following according to the guess made:

- *Great!! You guessed it right! You have made X guesses.*
- *Your number was too low. Please try again!*
- *Your number was too high. Please try again!*

If the player does not succeed in guessing the number this round, then the program will prompt the player to guess it again as below(replace Y with the guess number):

Try No.Y, please input the number:

If the player's guess is incorrect, your program should help the player to zero in on the correct answer by repeating the hints until the player finally gets the number right. At the end, display the number of guesses in total the player has made (in the text above, replace X with the proper number of guesses the player has made in total).

To show that your program works correctly, play it once with the upper bound 200 (i.e. range from 1 to 200) and submit the output as your script file (`guess.script`). Please compile your C code using **-ansi -Wall** options as below to specify ANSI code with all warnings:

```
gcc -o guess -ansi -Wall guess.c
```

You should submit your program code as file **guess.c**, a text file **guess.txt** briefly explaining how you designed your program, and a typescript **guess.script** which shows that you compile your program and run it. Try to guess a number between 1 to 200 (set the upper bound to 200).

HINT
To generate the initial random number, you have to use a random number generator which is provided by the C standard function **rand()**. This function generates a random number of type int in the range of 0 to 32767. This function is provided in the header file `stdlib.h`.

In practice, no computer function can produce truly random data – they only produce pseudo-random numbers. These are computed from a formula and the number sequences they produce are repeatable. A seed value is usually used by the random number generator to generate a number. Therefore, if you use the same seed value all the time, the same sequence of "random" numbers will be generated (i.e. your program will always produce the same "random" number in every program run). To avoid this, we can use the current time of the day to set the random seed, as this will always be changing with every program run. With this trick, your program will produce different guesses every time you run it.

To set the seed value, you have to use the function **srand()**, which is also defined in header file `stdlib.h`. For the current time of the day, you can use the function **time()**, which is defined in header file time.h (`stdlib.h` and `time.h` are header files just like the stdio.h file that we have been using so far).

In summary, use the following code fragments to generate the random number for the game:
1. Include the `stdlib.h` and `time.h` header files at the beginning of your program:

```
#include ⟨stdlib.h⟩
#include ⟨time.h⟩
```

2. Include the following lines at the beginning of your main function after the player inputs the upper bound *n*:

```
/* initialize the random number generator with the current time */
srand( time( NULL ) );

/* generate the random number in the range 0 to (n-1) */
/* randomNumber is an integer variable.  */
/* You need to define it at the beginning of the main function*/
randomNumber = rand() % n;
```

Here, `n` specifies the upper bound of the range in which the random number will be generated, and `randomNumber` is the integer variable which is assigned the generated random number.

2 Bonus: Maximum Steps of Guessing [5 points]

If you play this guessing game in a smart way, you can guess the right number in at most *N* steps. *N* here is the optimum maximum steps you need to guess a number in the range of [1-n]. Please give the equation for this maximum number of guessing steps in **guess.txt**.

HINT: the answer is an equation of the upper bound *n*.

3 A menu-driven calculator for floating point numbers [30 points]

The goal of this assignment is to practice using functions in C programming by performing arithmetic operations on double precision floating point numbers.

3.1 The Option Menu

Your program should be a menu driven program. At the start of your program, the user is prompted to input a floating point number. For instance, it may look like this:

```
Welcome to my floating point number calculator!
Please input a floating point number: 4.2
```

Then the program should display the current result and a menu. In our example, it looks like this:

```
----------------------------------------
The current result is: 4.200000
1. Add a floating point number to the current result;
2. Subtract a floating point number from the current result;
3. Multiply the current result by a floating point number;
4. Divide the current result by a floating point number;
5. Take the absolute value of the current result;
6. Quit

Please enter a selection:
```

Your program should let the user select an option, 1 through 6. If the user selects 6, your program simply exits. For other selections, your program should perform the corresponding floating point number operations.

When the user selects an operation, first ask the user to input another floating point number as an operand (with the exception of the reciprocal and quit operations). Once the user inputs the operand, your program should perform the required operation and display the current result and the option menu again.
In our example, the user selects 1, the add operation. So the program should ask the user to input a floating number to be added to 4.2, like this:

```
Please input a floating point number operand: 1.8
```

Then the program should bring up the current result and menu again. This time the current result should display 6.000000 (4.2 + 1.8 = 6.000000).

```
----------------------------------------
The current result is: 6.000000
1. Add a floating point number to the current result;
2. Subtract a floating point number from the current result;
3. Multiply the current result by a floating point number;
4. Divide the current result by a floating point number;
5. Take the absolute value of the current result;
6. Quit

Please enter a selection:
```

3.2 Printing the Current Value

The current floating point number always holds the result of the previous operation. At the beginning, there is no previous operation, so the current result just shows the floating point number you input at the start. In our example, it is 4.2.

3.3 Implementation

To implement the program, you need to define a set of functions; each function does a specific job, such as addition, subtraction, and division etc.

3.3.1 Function Declarations

As a starting point, use the following function declarations:

```
/*print a floating point number on console screen*/
void printNumber(double a);

/*accept user input*/
void getInput();

/*Add two floating point number return the value*/
double Add(double op1, double op2);

/*Subtract two floating point number return the value*/
double Subtract(double op1, double op2);

/*Multiply two floating point number return the value*/
double Multiply(double op1, double op2);

/*Divide two floating point number return the value */
/*Assume that the divisor is not zero */
double Divide(double op1, double op2);

/*return the absolute value of a floating point number*/
double Absolute(double a);
```

Functions `Add()`, `Subtract()`, `Multiply()`, and `Divide()` require two integers as their input parameters as the first and second operand for these binary operations. `printNumbers()` and `Absolute()` requires one input parameters because these will work only on the current number.

3.3.2 Global Variables

We will store the current floating point number as global variables, as follows:

```
double currValue;
```

This way, all of your functions in the program have access to the current number.

In addition, it will be helpful to have the number that the user inputs as a global variable. We recommend the following to store the second operand provided by the user. And the `getInput()` can be use to get the input from the user.

```
double newValue;
```

3.3.3 Error handling

A robust program is able to handle all situations, and reports any problems to the user if an error occurs. To get full credit, your program should be able to handle the following error:

- Notify the user with **`ERROR: Division by zero!`** if the user selects the division operation and inputs zero as the operand, and reprompt the user until a proper operand is entered.

3.4 What to submit

You need to save your program as *calculator.c*.

The way you demonstrate your code will depend on whether or not you completed the extra credit. If you did not complete the extra credit, perform the following steps to generate the script file:

1. Compile and run your program
2. Input the floating point number 4.2
3. Add the current result with 1.8
4. Subtract 2.71 from the current result
5. Multiply the current result by -10
6. Divide the current result by 32.9
7. Take the absolute value
8. Divide the current result by 0 /*error handling*/
9. Divide the current result by 0.333333
10. Exit your program

For your reference, the result obtained after step 9 should be 3.000003.

Name the script file *calculator.script*. If you forgot how to create a script file, check the instructions for homework 1.

You also need to create a *calculator.txt* file, which briefly explains your program.

4 Submission

Submission for the files will be similar to last weeks assignment. The only difference is that you need to create a directory called hw3/. Put all the files for assignment 3 in that directory and run the **/ecelib/bin/turnin10** command in the parent directory of hw3/ to submit your homework.

Note: please pay attention to any announcements on the course noteboard.