EECS 10: Assignment 5

October 26, 2007

Due Monday 11/5/2007 12:00pm

1 Roulette [20 points]

Write a program to simulate the casino game Roulette. We will use a simplified version of roulette. We will only use numbers from 0 to 36 (00 is excluded). Further, the user is only allowed to bet on odd or even numbers.
At the start, your program will prompt the user to enter the money she/he has (*balance*) and the *betting type*: *1-Even number, 2-Odd number, 0 to quit the game*. Then your program will prompt for the *betting amount.*

Following this, your program will generate a random number between 0 and 36 (including 0 and 36). Depending on the *betting type* chosen by the user, the program will check the generated number for *even or odd number*. If won, the user gets the *betting amount* which will be added to the *balance*. If lost, the user will lose the *betting amount* which will be deducted from the *balance.*

As long as the user has some *balance*, the program will repeat prompting the user for the next betting. Your program will quit if the user loses all the money (i.e. *balance* is 0), or when the user enters 0 when prompted for the *betting type*. When run, your program should look as follows:

```
Entering the casino, how much money do you have? $100
We are playing Roulette, odd or even bets only.
Place your bet!
Enter 1 for odd, 2 for even, 0 to quit: 1
How much money you want to bet? $20
You bet $20.00 on odd numbers.
The winning number is 23!
You Win!
Your balance is $120.00!!
Place your bet!
Enter 1 for odd, 2 for even, 0 to quit: 2
How much money you want to bet? $100
You bet $100.00 on even numbers.
The winning number is 29!
You Lose!
Your balance is $20.00!!
Place your bet!
Enter 1 for odd, 2 for even, 0 to quit: 0
You exit the casino with $20.00
```

What to turn in:

You should submit your program code as file **`roulette.c`**, a text file **`roulette.txt`** briefly explaining how you designed your program, and a typescript **`roulette.script`** which shows that you compile your program and run it for 5 bets.

HINT

For generating the 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 functions 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 provided by 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:

```
/* initialize the random number generator with the current time */
srand(time(0));
```

3. In your program, use the following to generate the random number:

```
/* generate the random number in the range 0 to 36 */
randomNumber = rand() % 37;
```

Here, **`randomNumber`** is the integer variable which is assigned the generated random number.

2 Square root approximation [20 points + 5 points (extra credit)]

Write a program to calculate the square root of any positive floating-point value.
At the beginning, the program should ask the user to input a positive number **`N`** in the range between 0 and 10000.

```
Please input a positive number (1 to 10000):
```

We will use a binary search approximation technique for this assignment. In particular, the program will always keep a range of a left bound **`L`** and a right bound **`R`**, where the actual square root **`S`** lies somewhere between **`L`** and **`R`**:

`L ≤ S ≤ R`

Consequently, it follows that **`L*L ≤ N=S*S ≤ R*R`**. Thus, to find **`S`**, we can compare **`L*L`** or **`R*R`** with **`N`**. The binary approximation then works as follows. First, we compute a value **`M`** that lies in the middle between the left bound **`L`** and the right bound **`R`**: **`M`** = **`L`**+(**`R`**-**`L`**)/2. Then, if **`M*M`** is less than **`N`**, the square root obviously lies somewhere in the right half of the current range (i.e. within **`M`** to **`R`**), otherwise in the left half of the current range (i.e. within **`L`** to **`M`**). The program then can use the proper half of the range as the new range and repeat the whole process. With each iteration, the search range is effectively reduced to half its previous size. Because of this, this technique is called binary search.
To start the search, we will use the range from **`0`** to **`N`** (which is guaranteed to contain the square root of **`N`**). We will stop the iteration, once we have reached a range that is smaller than **`0.0000000001`** so that we reach a precision of 10 digits after the decimal point for our approximation. (**HINT**: We will need **long double** variables for all variables in order to achieve this precision.)

The pseudo-code of the algorithm can be written as follows:

Start with a range of 0 to N
As long as the range is not accurate enough, repeat the following steps:
    Compute the middle of the range
    Compare the square of the middle value with N

If the middle value is less than the square root
Use middle-to-right as the new range
Otherwise
Use left-to-middle as the new range
Output the middle of the latest range as result

For example, to compute the square root of 10, the program will start with 5, which is in the middle between 0 and 10. Since 5*5=25 is larger than 10, the program will try the middle number 2.5 of left bound (0 to 5). Thus, the program compares 2.5*2.5 with 10. Because the result 6.25 is smaller than 10, it will pick 3.75 (the middle number of 2.5 and 5) as the next guess. By picking the middle number every time and comparing its square with the original number, the program gets closer to the actual square root.

To demonstrate the approximation procedure, your program should print the approximated square root in each iteration, as follows:

```
Please input a positive number (1 to 10000): 42
Please input a positive number (1 to 10000): 42
Iteration 1: the square root of 42.0000000000 is approximately 21.0000000000
Iteration 2: the square root of 42.0000000000 is approximately 10.5000000000
Iteration 3: the square root of 42.0000000000 is approximately 5.2500000000
...
Iteration 39: the square root of 42.0000000000 is approximately 6.4807406985
```

Note that your program should run properly for any real number between 1.0 and 10000.0 (not only for the demo value 42).

What to turn in:

You should submit your program code as file **root.c**, a text file **root.txt** briefly explaining how you designed your program, and a typescript **root.script** which shows that you compile your program and run it using the 42 as input.

For 5 extra credits:

Improve your program so that it can calculate the **n**-th root of any value. The value **n** should be a positive integer input by the user.

For example:

```
Please input a positive number (1 to 10000): 42
Please input the value of integer n (n>0): 5
Iteration 1: the 5th root of 42.0000000000 is approximately 21.0000000000
Iteration 2: the 5th root of 42.0000000000 is approximately 10.5000000000
Iteration 3: the 5th root of 42.0000000000 is approximately 5.2500000000
...
Iteration 39: the 5th root of 42.0000000000 is approximately 2.1117857650
```

To submit, use the same files as in Part 2, i.e. **root.c**, **root.txt**, and **root.script**. The script file should show the output of your program when the user inputs 42 and n=5.

3 Submission

Submission for the files will be similar to last week's assignment. Create a directory called hw5. Put all the files for assignment 5 in that directory and run the **/ecelib/bin/turnin** command to submit your homework.