Three Zebras

Keir Robertson

Loopy Zebras

In this quest, you get to play with looping. You will create a bunch of functions, each of which does something that requires you to iterate over a sequence of steps more than once.

Each of these functions is worth a varying number of points. It's up to you to find out how many points you can score by making the most progress you can.

As usual, I will give you template code you can copy and flesh out.

Important note

In miniquests where you have to calculate the value of some fractional quantity (e^x and gp terms) you may not use the Math library methods (e.g. sqrt, pow, etc). If you do, you'll find that your results may be very slightly different from mine. They have to match exactly for you to pass the miniquests. There's the challenge.

Your first miniquest - Guess it

To get past this checkpoint you must correctly implement the following function in the template code:

```
bool play_game(int n);
```

When I invoke this method, I will supply it an integer parameter, n, which you must treat as a secret number.

Then you must give the user 6 chances to guess the number using the exact script below:

As soon as the function begins, print the following on the console to greet the user:

```
Welcome to my number guessing game
```

followed by exactly two blank newlines.

Then your function must loop **at most** six times. During each iteration it must ask the user to guess the secret by printing the following exact prompt:

```
Enter your guess: 
```

Note - there is one space following the colon at the end of the prompt. You must not end the prompt with a newline.

Then it must confirm what it read back to the user in a new line as follows:

```
You entered: NUMBER
```

where *NUMBER* should be replaced by the value you read from the user.

Now you will compare the user's guess to your secret number. If they are equal, you must return the value `true` to the caller after printing the following message to the console:

```
You found it in N guess(es).
```

where you would replace the *N* with the actual number of guesses it took the user to find the number. This should be followed by exactly one newline.

If the user's guess is not equal to the secret number, you must simply loop back to the top and repeat the whole sequence.

Except, of course, if you've already done it six times. In that case, you must print a blank line, and the following message (two lines) on the console. Then return `false` to your caller.

```
I'm sorry. You didn't find my number.
It was SECRET
```

You should replace *SECRET* with the actual secret number.

This shouldn't require more than 20-25 lines of code total (to give you a rough idea of when you're unknowingly over-coding). Refer to the provided code template for more information.

Important: Your user input mechanism must be robust. I may try and break it by inputting a string when an integer is expected. In this case, your program should treat it as a 0, rather than break out of the logic.

You will find it easiest to insulate `cin` from corruption by reading into a string using `getline()` and using `istringstream` to extract an integer from it. Look up or ask how to do it.

None of the remaining functions you write in this quest will require user input.

Your second miniquest - Etox

You must implement:

```
double etox(double x, size_t n);
```

This is simply an extension of the `etox()` function you wrote in a previous quest. The difference is that this function takes two parameters: x has the same meaning as before. The new parameter, n, is the number of terms you will use from the summation to calculate the required value. To recap, `etox(x, n)` calculates e^x using the formula:

$$e^x = 1 + x + x^2/2! + x^3/3! + . . . x^{n-1}/(n-1)!$$

where `n!` is the factorial of n, which is the product of all positive integers at most equal to `n`.

Your third miniquest - Char counts

You must implement:

```
size_t count_chars(string s, char c);
```

This function must return the number of occurrences of the given character, c, in the string, s. 'Nuff said.

Your fourth miniquest - Greatest Common Divisor

You must implement Euclid's popular GCD finding algorithm through the function:

```
size_t gcd(size_t a, size_t b);
```

This function must return the GCD of the two given non-negative integers, a and b.

Your fifth miniquest - Terms of an AP

You must implement the function:

string get_ap_terms(int a, int d, size_t n);

which returns a string containing the first n terms of the arithmetic progression (AP) as a sequence of comma-separated values.

Recall that an AP is specified by two terms a and d. The first term is a, and the d is how much you add to each term to get the next term. So the first N terms of the above AP will be:

```
a, a+d, a+2d, . . . , a+(n-1)*d
```

Suppose, for example, that a = 1, d = 3, and n = 5. Then this function should return the string `"1,4,7,10,13"`

Note some important things about the returned value: There are no newlines or spaces. Only commas separate the numbers. There is NO comma before the first number or after the last number. The format has to match exactly.

Your sixth miniquest - Terms of a GP

Copy and adapt the function in the previous mini-quest to turn it into a function that returns the terms of a geometric progression, rather than an arithmetic one. We're talking about changes to a couple of lines.

However, note that the parameter types are different. You should operate in `double` space to keep high precision (not in `integer` space).

Recall that a GP is specified by two terms a and r. The first term of the sequence is a, and the r is by how much multiply each term to get the next term. So the first N terms of the above AP will be:

`a, ar ar`^{2}`, . . . , ar`$^{n-1}$

Suppose, for example, that a = 4, r = 0.5, and n = 6. Then this function should return the string `"4,2,1,0.5,0.25,0.125"`

Your seventh miniquest - Fibonacci

Implement the following function:

```
double get_nth_fibonacci_number(size_t n);
```

which returns the nth number in the Fibonacci sequence. The first two terms of the sequence are both 1. Thereafter successive terms are simply the sums of their two previous terms. Thus the sequence goes:

```
1, 1, 2, 3, 5, 8, 13, 21, . . .
```

Starter code

First your header file:

```
//
//  Looping_Functions.h
//
// This is your header file. No need to make any major changes. You can just feel free
// to copy it verbatim.

#ifndef Looping_Functions_h
#define Looping_Functions_h

// Declarations of the functions in looping_functions.cpp

bool play_game(int n);
double etox(double x, size_t n);
size_t count_chars(std::string s, char c);
size_t gcd(size_t n1, size_t n2);
std::string get_ap_terms(int a, int d, size_t n);
std::string get_gp_terms(double a, double r, size_t n);
double get_nth_fibonacci_number(size_t n);

#endif /* Looping_Functions_h */
```

And then your cpp file:

```
//  Student ID: 12345678
//  TODO - Replace the number above with your actual Student ID
//
//  Looping_Functions.cpp
//
//  Created by Anand Venkataraman on 8/9/19.
//  Copyright © 2019 Anand Venkataraman. All rights reserved.
//

#include <iostream>
#include <sstream>

using namespace std;

// Give the user 6 chances to guess the secret number n (0-10). If they get it,
// say so and return true. Else say so and return false.
bool play_game(int n) {
    // TODO - Your code here
}

// Calculate e^x using the series summation up to exactly the first
// n terms including the 0th term.
double etox(double x, size_t n) {
    // TODO - Your code here
}

// Return the number of occurrences of char c in string s
size_t count_chars(string s, char c) {
    // TODO - Your code here
}
```

```
// Use Euclid's algorithm to calculate the GCD of the given numbers
size_t gcd(size_t a, size_t b) {
    // TODO - Your code here

}

// Return a string of the form n1,n2,n3,... for the given AP.
string get_ap_terms(int a, int d, size_t n) {
    // TODO - Your code here
}

// Return a string of the form n1,n2,n3,... for the given GP.
string get_gp_terms(double a, double r, size_t n) {
    // TODO - Your code here
}

double get_nth_fibonacci_number(size_t n) {
    // TODO - Your code here
}
```

Testing your own code

You should test your functions using your own `main()` method in which you try and call your functions in many different ways and cross-check their return values against your hand-computed results. But when you submit you must NOT submit your main method. I will use my own and invoke your functions in many creative ways. Hopefully you've thought of all of them.

Submission

When you think you're happy with your code and it passes all your own tests, it is time to see if it will also pass mine.

1. Head over to https://quests.nonlinearmedia.org
2. Enter the secret password for this quest in the box.
3. Drag and drop your `Looping_Functions.*` files into the button and press it. Make sure your filename is exactly as above.
4. Wait for me to complete my tests and report back (usually a minute or less).

Points and Extra Credit Opportunities

I monitor the discussion forums closely and award extra credit points for well-thought out and helpful discussions.

May the best coders win. That may just be all of you.

Happy Hacking,

&