5.x — Chapter 5 comprehensive quiz

BY ALEX ON JULY 16TH, 2015 | LAST MODIFIED BY ALEX ON MARCH 22ND, 2018

Quick review

If statements allow us to execute a statement based on whether some condition is true. *Else statements* execute if the associated if statement is false. You can chain together multiple if and else statements.

Switch statements provide a cleaner and faster method for selecting between a number of discrete items. Switch statements pair great with enumerations.

Goto statements allow the program to jump to somewhere else in the code. Don't use these.

While loops allow the program to loop as long as a given condition is true. The condition is evaluated before the loop executes.

Do while loops are the same as while loops, but the condition is evaluated after the loop execution. They're great for menus or things that need to execute at least once.

For loops are the most used loop, and are perfect when you need to loop a specific number of times.

Break statements allow us to break out of a *switch*, *while*, *do while*, or *for* loop. Or a *for each* loop, which we haven't covered yet.

Continue statements allow us to move immediately to the next loop iteration. Be careful when using these with while and do while loops, as your loop counter may not get incremented properly.

And finally, random numbers give us a way to make our programs behave different each time they are run. We'll see an example of this in the quiz below!

Quiz time!

Warning: The quizzes start getting harder from this point forward, but you can do it. Let's rock these quizzes!

1) In the chapter 2 comprehensive quiz, we wrote a program to simulate a ball falling off of a tower. Because we didn't have loops yet, the ball could only fall for 5 seconds.

Take the program below and modify it so that the ball falls for as many seconds as needed until it reaches the ground.

In constants.h:

```cpp
#ifndef CONSTANTS_H
#define CONSTANTS_H

namespace myConstants
{
    const double gravity(9.8); // in meters/second squared
}
#endif
```

In your main code file:

```cpp
#include <iostream>
#include "constants.h"

// gets initial height from user and returns it
double getInitialHeight()
{
    std::cout << "Enter the height of the tower in meters: ";
    double initialHeight;
    std::cin >> initialHeight;
    return initialHeight;
}

// Returns height from ground after "secondsPassed" seconds
```

```
double calculateHeight(double initialHeight, int secondsPassed)
{
    // Using formula: [ s = u * t + (a * t^2) / 2 ], here u(initial velocity) = 0
    double distanceFallen = (myConstants::gravity * secondsPassed * secondsPassed) / 2;
    double currentHeight = initialHeight - distanceFallen;

    return currentHeight;
}

// Prints height every second till ball has reached the ground
void printHeight(double height, int secondsPassed)
{
    if (height > 0.0)
    {
        std::cout << "At " << secondsPassed << " seconds, the ball is at height:\t" << height <<
            " meters\n";
    }
    else
        std::cout << "At " << secondsPassed << " seconds, the ball is on the ground.\n";
}

void calculateAndPrintHeight(double initialHeight, int secondsPassed)
{
    double height = calculateHeight(initialHeight, secondsPassed);
    printHeight(height, secondsPassed);
}

int main()
{
    const double initialHeight = getInitialHeight();

    calculateAndPrintHeight(initialHeight, 0);
    calculateAndPrintHeight(initialHeight, 1);
    calculateAndPrintHeight(initialHeight, 2);
    calculateAndPrintHeight(initialHeight, 3);
    calculateAndPrintHeight(initialHeight, 4);
    calculateAndPrintHeight(initialHeight, 5);

    return 0;
}
```

Hide Solution

In constants.h:

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

namespace myConstants
{
    const double gravity(9.8); // in meters/second squared
}
#endif
```

In your main code file:

```
#include <iostream>
#include "constants.h"

// gets initial height from user and returns it
double getInitialHeight()
{
    std::cout << "Enter the height of the tower in meters: ";
    double initialHeight;
    std::cin >> initialHeight;
    return initialHeight;
}
```

```cpp

// Returns height from ground after "seconds" seconds
double calculateHeight(double initialHeight, int seconds)
{
    // Using formula: [ s = u * t + (a * t^2) / 2 ], here u(initial velocity) = 0
    double distanceFallen = (myConstants::gravity * seconds * seconds) / 2;
    double currentHeight = initialHeight - distanceFallen;

    return currentHeight;
}

// Prints height every second till ball has reached the ground
void printHeight(double height, int seconds)
{
    if (height > 0.0)
        std::cout << "At " << seconds << " seconds, the ball is at height: " << height << " meters\n";
    else
        std::cout << "At " << seconds << " seconds, the ball is on the ground.\n";
}

int main()
{
    const double initialHeight = getInitialHeight();

    int seconds = 0;
    double height;

    do
    {
        height = calculateHeight(initialHeight, seconds);
        printHeight(height, seconds);
        ++seconds;
    } while (height > 0.0);

    return 0;
}
```

2a) Implement a game of hi-lo. First, your program should pick a random integer between 1 and 100. The user is given 7 tries to guess the number.

If the user does not guess the correct number, the program should tell them whether they guessed too high or too low. If the user guesses the right number, the program should tell them they won. If they run out of guesses, the program should tell them they lost, and what the correct number is. At the end of the game, the user should be asked if they want to play again. If the user doesn’t enter ‘y’ or ‘n’, ask them again.

Note: You do not need to implement error handling for the user’s guess.

Here’s what your output should look like:

```
Let's play a game.  I'm thinking of a number.  You have 7 tries to guess what it is.
Guess #1: 64
Your guess is too high.
Guess #2: 32
Your guess is too low.
Guess #3: 54
Your guess is too high.
Guess #4: 51
Correct! You win!
Would you like to play again (y/n)? y
Let's play a game.  I'm thinking of a number.  You have 7 tries to guess what it is.
Guess #1: 64
Your guess is too high.
Guess #2: 32
Your guess is too low.
Guess #3: 54
```

```
Your guess is too high.
Guess #4: 51
Your guess is too high.
Guess #5: 36
Your guess is too low.
Guess #6: 45
Your guess is too low.
Guess #7: 48
Your guess is too low.
Sorry, you lose.  The correct number was 49.
Would you like to play again (y/n)? q
Would you like to play again (y/n)? f
Would you like to play again (y/n)? n
Thank you for playing.
```

Hints:
* Seed your random number generator with time(0).
* Visual Studio users: Due to a flaw in the Visual Studio implementation of rand(), call rand() once after seeding to discard the first result.
* Use the getRandomNumber() function from lesson **5.9 -- Random number generation** to pick a random number.
* Write a function that allows the user to play a single game of hi-lo.
* Write a function that asks the user if they want to play again and handles the looping logic for an incorrect input.

Hide Solution

```cpp
#include <iostream>
#include <cstdlib> // for srand() and rand()
#include <ctime> // for time()

// Generate a random number between min and max (inclusive)
// Assumes srand() has already been called
int getRandomNumber(int min, int max)
{
    static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);  // static used for efficiency, so we only calculate this value once
    // evenly distribute the random number across our range
    return static_cast<int>(rand() * fraction * (max - min + 1) + min);
}

// returns true if the user won, false if they lost
bool playGame(int guesses, int number)
{
    // Loop through all of the guesses
    for (int count = 1; count <= guesses; ++count)
    {
        std::cout << "Guess #" << count << ": ";
        int guess;
        std::cin >> guess;

        if (guess > number)
            std::cout << "Your guess is too high.\n";
        else if (guess < number)
            std::cout << "Your guess is too low.\n";
        else // guess == number
            return true;
    }

    return false;
}

bool playAgain()
{
    // Keep asking the user if they want to play again until they pick y or n.
    char ch;
    do
    {
        std::cout << "Would you like to play again (y/n)? ";
        std::cin >> ch;
    } while (ch != 'y' && ch != 'n');

    return (ch == 'y');
}

int main()
{
    srand(static_cast<unsigned int>(time(0))); // set initial seed value to system clock
    rand(); // discard the first result because rand() doesn't randomize the first random number very well in Visual Studio

    const int guesses = 7; // the user has this many guesses

    do // while the user still wants to play
    {
        int number = getRandomNumber(1, 100); // this is the number the user needs to guess

        std::cout << "Let's play a game.  I'm thinking of a number.  You have " << guesses << " tries to guess what it is.\n";

        bool won = playGame(guesses, number);
        if (won)
            std::cout << "Correct!  You win!\n";
        else
            std::cout << "Sorry, you lose.  The correct number was " << number << "\n";

    }
    while (playAgain());

    std::cout << "Thank you for playing.\n";
    return 0;
}
```

2b) Update your previous solution to handle invalid input (e.g. 'x') or valid input with extraneous characters (e.g. "43x") when the user is guessing a number.

Hint: Write a separate function to handle the user inputting their guess (along with the associated error handling).

Hide Solution

```
#include <iostream>
#include <cstdlib> // for srand() and rand()
#include <ctime> // for time()

// Generate a random number between min and max (inclusive)
// Assumes srand() has already been called
int getRandomNumber(int min, int max)
{
    static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);  // static used for efficiency, so we only calculate this value once
                                                                                  // evenly distribute the random number across our range
    return static_cast<int>(rand() * fraction * (max - min + 1) + min);
}

int getGuess(int count)
{
    while (1) // loop until user enters valid input
    {
        std::cout << "Guess #" << count << ": ";
        int guess;
        std::cin >> guess;
```

```cpp

        if (std::cin.fail()) // has a previous extraction failed?
        {
            // yep, so let's handle the failure
            std::cin.clear(); // put us back in 'normal' operation mode
            std::cin.ignore(32767, '\n'); // and remove the bad input
        }
        else
        {
            std::cin.ignore(32767, '\n'); // remove any extraneous input
            return guess;
        }
    }
}

// returns true if the user won, false if they lost
bool playGame(int guesses, int number)
{
    // Loop through all of the guesses
    for (int count = 1; count <= guesses; ++count)
    {

        int guess = getGuess(count);

        if (guess > number)
            std::cout << "Your guess is too high.\n";
        else if (guess < number)
            std::cout << "Your guess is too low.\n";
        else // guess == number
            return true;
    }

    return false;
}

bool playAgain()
{
    // Keep asking the user if they want to play again until they pick y or n.
    char ch;
    do
    {
        std::cout << "Would you like to play again (y/n)? ";
        std::cin >> ch;
    } while (ch != 'y' && ch != 'n');

    return (ch == 'y');
}

int main()
{
    srand(static_cast<unsigned int>(time(0))); // set initial seed value to system clock
    rand(); // discard the first result because rand() doesn't randomize the first random number very well in Visual Studio

    const int guesses = 7; // the user has this many guesses

    do // while the user still wants to play
    {
        int number = getRandomNumber(1, 100); // this is the number the user needs to guess

        std::cout << "Let's play a game.  I'm thinking of a number.  You have " << guesses << " tries to guess what it is.\n";

        bool won = playGame(guesses, number);
        if (won)
            std::cout << "Correct!  You win!\n";
        else
```

```
            std::cout << "Sorry, you lose.  The correct number was " << number << "\n";

    } while (playAgain());

    std::cout << "Thank you for playing.\n";
    return 0;
}
```

Share this:

Facebook Twitter Google Pinterest

C++ TUTORIAL | PRINT THIS POST

336 comments to 5.x — Chapter 5 comprehensive quiz

« Older Comments 1 … 3 4 5

Adam

July 15, 2018 at 7:05 pm · Reply

Hi, I'm pretty new to coding and my code looks slightly different from yours but I'm still able to get the same result. I'm not sure if my code is poorly written and I should try to make it look more similar to yours or just leave it. One quick question I have is how do I know if my code is inefficient so I know what parts should be changed? Thanks in advance.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

int getRandomNumber()
{
    int min{1};
    int max{RAND_MAX};
    static const double fraction = 1.0 / (RAND_MAX + 1.0);  // static used for efficiency, so we onl
y calculate this value once
    // evenly distribute the random number across our range
    return min + static_cast<int>((max - min + 1) * (std::rand() * fraction));
}
int checkInput(int guessnum, int yourguess)
{
    while (true) // Loop until user enters a valid input
    {
        std::cout << "Guess #" << guessnum << ": ";
        std::cin >> yourguess;
        std::cin.ignore(32767, '\n');
        if (std::cin.fail()) // has a previous extraction failed?
        {
            // yep, so let's handle the failure
```