7.x — Chapter 7 summary and quiz

ALEX JUNE 28, 2021

Quick review

The specific sequence of statements that the CPU executes in a program is called the program's execution path. A straight-line program takes the same path every time it is run.

Control flow statements (also called Flow control statements) allow the programmer to change the normal path of execution. When a control flow statement causes the program to begin executing some non-sequential instruction sequence, this is called a branch.

A conditional statement is a statement that specifies whether some associated statement(s) should be executed or not.

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

A dangling else occurs when it is ambiguous which `if statement` an `else statement` is connected to. `Dangling else` statements are matched up with the last unmatched `if statement` in the same block. Thus, we trivially avoid `dangling else` statements by ensuring the body of an `if statement` is placed in a block.

A null statement is a statement that consists of just a semicolon. It does nothing, and is used when the language requires a statement to exist but the programmer does not need the statement to do anything.

Switch statements provide a cleaner and faster method for selecting between a number of matching items. Switch statements only work with integral types. Case labels are used to identify the values for the evaluated condition to match. The statements beneath a default label are executed if no matching case label can be found.

When execution flows from a statement underneath a label into statements underneath a subsequent label, this is called fallthrough. A `break statement` (or `return statement`) can be used to prevent fallthrough. The [[fallthrough]] attribute can be used to document intentional fallthrough.

Goto statements allow the program to jump to somewhere else in the code, either forward or backwards. These should generally be avoided, as they can create spaghetti code, which occurs when a program has a path of execution that resembles a bowl of spaghetti.

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

An infinite loop is a loop that has a condition that always evaluates to `true`. These loops will loop forever unless another control flow statement is used to stop them.

A loop variable (also called a counter) is an integer variable used to count how many times a loop has executed. Each execution of a loop is called an iteration.

Do while loops are similar to while loops, but the condition is evaluated after the loop executes instead of before.

For loops are the most used loop, and are ideal when you need to loop a specific number of times. An off-by-one error occurs when the loop iterates one too many or one too few times.

Break statements allow us to break out of a switch, while, do while, or for loop (also `range-based for loops`, which we haven't covered yet). Continue statements allow us to move immediately to the next loop iteration.

Halts allow us to terminate our program. Normal termination means the program has exited in an expected way (and the `status code` will indicate whether it succeeded or not). std::exit() is automatically called at the end of `main`, or it can be called explicitly to terminate the program. It does some cleanup, but does not cleanup any local variables, or unwind the call stack.

Abnormal termination occurs when the program encountered some kind of unexpected error and had to be shut down. std::abort can be called for an abnormal termination.

Scope creep occurs when a project's capabilities grow beyond what was originally intended at the start of the project or project phase.

Software verification is the process of testing whether or not the software works as expected in all cases. A unit test is a test designed to test a small portion of the code (typically a function or call) in isolation to ensure a particular behavior occurs as expected. Unit test frameworks can help you organize your unit tests. Integration testing tests the integration of a bunch of units together to ensure they work properly.

Code coverage refers to how much of the source code is executed while testing. Statement coverage refers to the percentage of statements in a program that have been exercised by testing routines. Branch coverage refers to the percentage of branches that have been executed by testing routines. Loop coverage (also called the 0, 1, 2 test) means that if you have a loop, you should ensure it works properly when it iterates 0 times, 1 time, and 2 times.

The happy path is the path of execution that occurs when there are no errors encountered. A sad path is one where an error or failure state occurs. A non-recoverable error (also called a fatal error) is an error that is severe enough that the program can't continue running. A program that handles error cases well is robust.

A buffer is a piece of memory set aside for storing data temporarily while it is moved from one place to another.

The process of checking whether user input conforms to what the program is expecting is callednput validation.

std::cerr is an output stream (like `std::cout`) designed to be used for error messages.

A precondition is any condition that must always be true prior to the execution of some segment of code. An invariant is a condition that must be true while some component is executing. A postcondition is any condition that must always be true after the execution of some code.

An assertion is an expression that will be true unless there is a bug in the program. In C++, runtime assertions are typically implemented using the assert preprocessor macro. Assertions are usually turned off in non-debug code. A static_assert is an assertion that is evaluated at compile-time.

Assertions should be used to document cases that should be logically impossible. Error handling should be used to handle cases that are possible.

Quiz time

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

Question #1

In the chapter 4 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:

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

namespace myConstants
{
    inline constexpr double gravity { 9.8 }; // in meters/second squared
}
#endif
```

In your main code file:

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

double calculateHeight(double initialHeight, int seconds)
{
    double distanceFallen { myConstants::gravity * seconds * seconds / 2 };
    double heightNow { initialHeight - distanceFallen };

    // Check whether we've gone under the ground
    // If so, set the height to ground-level
    if (heightNow < 0.0)
        return 0.0;
    else
        return heightNow;
}

void calculateAndPrintHeight(double initialHeight, int time)
{
    std::cout << "At " << time << " seconds, the ball is at height: " << calculateHeight(initialHeight,
time) << "\n";
}

int main()
{
    std::cout << "Enter the initial height of the tower in meters: ";
    double initialHeight;
    std::cin >> initialHeight;

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

    return 0;
}
```

Show Solution

Question #2

A prime number is a natural number greater than 1 that is evenly divisible (with no remainder) only by 1 and itself. Complete the following program by writing the `isPrime()` function using a for-loop. When successful, the program will print "Success!".

```cpp
#include <iostream>
#include <cassert>

bool isPrime(int x)
{
    // write this function using a for loop
}

int main()
{
    assert(!isPrime(0));
    assert(!isPrime(1));
    assert(isPrime(2));
    assert(isPrime(3));
    assert(!isPrime(4));
    assert(isPrime(5));
    assert(isPrime(7));
    assert(!isPrime(9));
    assert(isPrime(11));
    assert(isPrime(13));
    assert(!isPrime(15));
    assert(!isPrime(16));
    assert(isPrime(17));
    assert(isPrime(19));
    assert(isPrime(97));
    assert(!isPrime(99));
    assert(!isPrime(99));
    assert(isPrime(13417));

    std::cout << "Success!";

    return 0;
}
```

Show Solution

Next lesson

8.1 Implicit type conversion (coercion)

Back to table of contents

Previous lesson

7.17 Assert and static_assert

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ coc

Name*

Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: POST COMMENT

DP N N FOUT

Newest