Outline

- In this lesson, we will:
  - Introduce the concept of breaking out of a loop
  - Modify a previous example to finish quicker
  - Look at how to break out of nested loops

Determining if an integer is prime

- Consider this program:

  ```cpp
  int main()
  { 
    int n;
    std::cout << "Enter an integer: ";
    std::cin >> n;
    bool is_prime=true;
    if ( n%2 == 0 ) 
    { 
      is_prime = false;
    } 
    else 
    { 
      for ( int k=3; k < n; k += 2 ) 
      { 
        if ( n%k == 0 ) 
        { 
          is_prime = false;
        } 
      } 
    } 
    return 0;
  }
  ```

```cpp
if ( is_prime ) { 
  std::cout << "The integer " << n << " is prime" << std::endl;
} else { 
  std::cout << "The integer " << n << " is not prime" << std::endl;
}
return 0;
```
Suppose you test if 303 is prime:

- After $k = 3$, we're done; we know that 303 is not prime...
  - So why test all other numbers?

Now, given an integer $n$ that is not prime, if $n = m_1 m_2$ then either:

1. If $n$ is a perfect square, it may be that $m_1 = m_2 = \sqrt{n}$
2. Otherwise, if $m_1 < \sqrt{n}$, then $m_2 > \sqrt{n}$

- For example, $\sqrt{420} \approx 20.4939$
- We see that, $\sqrt{420} = 2 \times 210 = 3 \times 140$
  = $4 \times 105$ = $5 \times 84$
  = $6 \times 70$ = $7 \times 60$
  = $10 \times 42$ = $12 \times 35$
  = $14 \times 30$ = $15 \times 28$
  = $20 \times 21$

Thus, an even better program is:

```java
if (nX2 == 0) {
    is_prime = false;
} else {
    for (int k(3); k < n; k += 2) {
        if (nXk == 0) {
            is_prime = false;
            break;
        }
        // Execution continues here...
    }
}
```

If $k * k > n$ and $nXk == 0$, then $n/k < k$, so we would have already tested it. Thus, we only need to test those $k$ less than or equal to the square root of $n$.

```java
if (k*k > n) {
    break;
}
```

// Execution continues here...

---

The `break` statement allows us to terminate a loop:

- The loop immediate stops:
  - The condition is not tested
  - The update statement is not executed
- Execution jumps to the end of the loop and continues from there

```java
if (nX2 == 0) {
    is_prime = false;
} else {
    for (int k(3); k < n; k += 2) {
        if (nXk == 0) {
            is_prime = false;
            break;
        }
        // Execution continues here...
    }
}
```
Ending a loop early

- Consider the benefits:
  - To test if a number around 1,000,000 is prime:
    - Previously, we tested approximately 500,000 numbers
    - Now we test at approximately 500
  - To test if a number around 100 million is prime:
    - Previously, we would have tested approximately 50 million numbers
    - Now we test at approximately 5000

Modifying the condition

- As a simple observation, we didn’t have to use a break statement, as both these conditions could have been added to the condition:
  ```cpp
  if ( n%2 == 0 ) {
      is_prime = false;
  } else {
      // Stop looping if we ever find is_prime == false or k > \sqrt{n}
      for ( int k=3; is_prime && (k*k <= n); k += 2 ) {
          if ( n%k == 0 ) {
              is_prime = false;
          }
      }
  }
  ```
  - Both work, both are acceptable approaches to solving this problem

Integer square root

- The integer square root of $n$ is the largest integer $m$ such that $m^2 \leq n$
- If $n$ is a perfect square, then $m^2 = n$ and the integer square root is $m$
  - For example, $10^2 = 100$
- Otherwise, consider 99:
  - $9^2 = 81 < 99$ and $10^2 = 100 > 99$,
    so the integer square root of 99 is 9

Centralized smart contract deployment

- Consider this program:
  ```cpp
  int main() {
      int n; std::cout << "Enter an integer: ";
      std::cin >> n;
      int isqrt{0};
      for ( int m{1}; m < n; ++m ) {
          if ( m*m <= n ) {
              isqrt = m;
          }
      }
      std::cout << "The integer square root of " << n << " is "
              << isqrt << std::endl;
      return 0;
  }
  ```
• One problem with this program is that we test all integers up to and including \( n - 1 \), even if the integer square root is much smaller...

\[
\text{for ( int m1; m1 < n; ++m1) }
\text{ if ( \text{m1}^2 \leq n )}
\text{ isqrt = m1; }
\]

• What happens if \( m^2 \leq n \) is false?
  – This must mean that \( m^2 > n \), in which case we are finished

Finding a sum of squares

• Suppose we want to find if \( n \) is the sum of two non-zero squares:
  – Is \( n = m_1^2 + m_2^2 \) for two non-zero integers \( m_1 \) and \( m_2 \)?
  – For many engineering problems, we only need to have one example
    • Thus, once we find one pair of integers, we're finished...

Finding a sum of squares

• Consider this program:

\[
\text{int main() }
\text{ { }
\text{ int n(); }
\text{ std::cout << “Enter an integer: “; }
\text{ std::cin >> n; }
\]

\[
\text{ bool is_found(false); }
\text{ for ( int m1[1]; m1 < n; ++m1) }
\text{ for ( int m2[1]; m2 < n; ++m2) }
\text{ if ( (m1*m1 + m2*m2) == n )}
\text{ is_found = true; break; }
\]

\[
// What are m1 and m2?
\text{ return 0; }
\}
Finding a sum of squares

Consider this program:

```cpp
text
```

```cpp
Finding a sum of squares

- We now try running our program:
  
Enter an integer: 121
  121 is not the sum of two non-zero squares

Enter an integer: 122
  122 = 122^2 + 122^2
```

Finding a sum of squares

- The break only exits the inner loop

```cpp
Finding a sum of squares

if ( is_found ) {
    std::cout << n << " is not the sum of two non-zero squares" << std::endl;
} else {
    std::cout << n << std::endl;
}
return 0;
```
Finding a sum of squares

• Solution
  – If we find a pair of integers that satisfies our condition, break out of the outer loop, as well
    
    ```java
    for ( m1 = 1; m1 < n; ++m1 ) {
      for ( m2 = m1; m2 < n; ++m2 ) {
        if ( (m1*m1 + m2*m2) == n ) {
          is_found = true;
          break; // exits the inner for loop
        }
      }
    }
    if ( is_found ) {
      break; // exits the outer for loop
    }
    ```

  • We now try running our program:

    Enter an integer: 122
    122 = 1^2 + 11^2

Summary

• Following this lesson, you now
  – Understand the purpose of a break statement
    • It ends the execution of a for loop
  – Know that the break statement is just break;
  – Understand how to finish a loop early if there is no need to continue executing the loop
    • It only jumps out of the loop in which it is found

References

https://en.wikipedia.org/wiki/Control_flow#Early_exit_from_loops

[2] cplusplus.com
http://www.cplusplus.com/doc/tutorial/control/
Acknowledgments

Proof read by Dr. Thomas McConkey and Charlie Liu.

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using Consolas.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see https://www.rbg.ca/ for more information.

Disclaimer

These slides are provided for the ECE 150 Fundamentals of Programming course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.