**While loops**

While loops are used to perform a task repeatedly while a certain condition is true. They are useful when the number of iterations is not known at the start of the loop.

### Outline

- In this lesson, we will:
  - See how to implement while loops in C++
  - Implement while loops that
    - Play a guessing game with the user
    - Find all prime factors of a given integer
    - Implement the Collatz conjecture
  - Learn how to convert a description of an algorithm to one that you can program
    - We will use the greatest-common divisor algorithm
  - Observe that all for loops can be written as while loops

### Repetition statements

- A for loop is a special case of a repetition statement:
  - The loop body is executed a fixed number of times based on
    - The initial value of a loop variable,
    - A condition involving the loop variable, and
    - An update to that loop variable executed after the loop body is run

```cpp
for ( int k=0; k < n; ++k ) {
    // Loop body
    std::cout << k << ", ";
}
```

- Very often, at compile time, you can determine how often this loop will be executed

- In some cases, however, we don't know how often a loop body will be executed
  - An alternative approach is a while loop
  - A while loop only has a condition and a loop body
    - The loop body is run as long as the condition is true

```cpp
while ( Boolean-valued condition ) {
    // The loop body or block of statements
    // - to be executed as long as the
      // condition is 'true'
}
```

- Continue executing here as soon as the
  // condition evaluates to 'false'
Infinite loops

• A common loop is the infinite loop:
  ```cpp
  while ( true ) {
      // The loop body or block of statements
      // - will be repeatedly executed forever
      //   or until we get out of the loop otherwise
  }
  ```

Accessing a value from the user

• Like with the for loop, a while loop can use a break statement:
  ```cpp
  int n{};
  std::cout << "Enter a positive integer: ";
  std::cin >> n;
  while ( n <= 0 ) {
      std::cout << "Enter a positive (> 0) integer: ";
      std::cin >> n;
  }
  ```

A guessing game

• Suppose we want to play a guessing game:
  – Player A enters a number to be guessed
  – Player B continues to try to guess that number until that Player B guesses correctly

• Put another way:
  – Inside an infinite loop:
    • Query Player B for a guess
    • If that guess is correct, we will break out of this loop
A guessing game

• A while loop is used when it is unknown how often a loop may run

```cpp
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
   int secret_number{};
   std::cout << "Player A: enter a secret number: ";
   std::cin >> secret_number;

   while (true) {
      int guessed_number{};
      std::cout << "Player B: enter a guess: ";
      std::cin >> guessed_number;

      if (guessed_number == secret_number) {
         std::cout << "You guessed the secret number" << std::endl;
         break;
      }
      else {
         std::cout << "Incorrect guess" << std::endl;
      }
   }
   return 0;
```

A guessing game

• Implementing this game

```cpp
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
   int secret_number{};
   int guessed_number{};
   std::cout << "Player A: enter a secret number from 1 to 100: ";
   std::cin >> secret_number;

   while (secret_number < 1 || secret_number > 100) {
      std::cout << "Enter a secret number from 1 to 100: ";
      std::cin >> secret_number;
   }
   return 0;
```

The game of high-low

• Let’s refine this guessing game so that
  – Player A enters a number between 1 and 100 to be guessed
  – Player B continues to try to guess that number
    • If the guess is correct, the game is over
    • If the guess is greater than the number, we tell the player that the guess is too high
    • Otherwise, we tell the player that the guess is too low

• Put another way:
  – In an infinite loop:
    • Query Player B for a guess
    • If that guess is correct, we will break out of this loop
    • Otherwise, we will tell the player if the guess was too high or too low
The game of high-low

• An alternative condition
  #include <iostream>
  // Function declarations
  int main();
  // Function definitions
  int main()
  {
    int secret_number{};
    std::cout << "Player A: enter a secret number from 1 to 100: ";
    std::cin >> secret_number;
    while ( !( secret_number >= 1 && secret_number <= 100 ) ) {
      std::cout << "Enter a secret number from 1 to 100: ";
      std::cin >> secret_number;
    }
    return 0;
  }

Finding prime factors of an integer

• Suppose we want to print all prime factors of an integer:
  ~ For example:
    • 123 = 3 × 41
    • 124 = 2 × 2 × 31
    • 125 = 5 × 5 × 5
  ~ Now, 2666 is divisible by 2, so the prime factors are:
    ~ 2 and the prime factors of 2666 ÷ 2 = 1333
  ~ This looks like an interesting strategy...

while ( true )
{
  int guessed_number{};
  std::cout << "Player B: enter a guess from 1 to 100: ";
  std::cin >> guessed_number;
  if ( guessed_number == secret_number ) {
    std::cout << "You guessed the secret number" << std::endl;
    break;
  }
  else if ( guessed_number < secret_number )
  {
    std::cout << "Too low, guess again..." << std::endl;
  }
  else
  {
    std::cout << "Too high, guess again..." << std::endl;
  }
}

Finding prime factors of an integer

int main() {
  int n;
  std::cout << "Enter a positive integer to be factored: ";
  std::cin >> n;
  int possible_factor{2};
  while ( n > 1 )
  {
    while ( n % possible_factor == 0 )
    {
      std::cout << possible_factor << ", ";
      n /= possible_factor;
    }
    // Ideally, we should go to the next highest prime,
    // but this works, too.
    ++possible_factor;
  }
  return 0;
}
Collatz conjecture

- The Collatz conjecture says that if you start with any positive integer \( n \) and
- If \( n \) is even, divide it by two
- If \( n \) is odd, multiply it by three and add one

- If ever \( n = 1 \), then the sequence carries on forever:
  \( 1, 4, 2, 1, 4, 2, 1, 4, 2, 1, \ldots \)

- The Collatz conjecture says that regardless of your initial \( n \), this sequence always gets to 1

Collatz conjecture

- Here are some longer examples
  - For example,
    \begin{itemize}
    \item 27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 515, 1546, 773, 2319, 1159, 579, 289, 448, 224, 112, 56, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
    \end{itemize}

Collatz conjecture

- We can try this with any number of initial values
  1
  2, 1
  3, 10, 5, 16, 8, 4, 2, 1
  4, 2, 1
  5, 16, 8, 4, 2, 1
  6, 3, 10, 5, 16, 8, 4, 2, 1
  7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
  8, 4, 2, 1
  9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
  10, 5, 16, 8, 4, 2, 1
  11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
  12, 6, 3, 10, 5, 16, 8, 4, 2, 1

Collatz conjecture

- We can implement this:

```cpp
int main() {
    int n;
    std::cout << "Enter a positive integer: ";
    std::cin >> n;
    while (n != 1) {
        std::cout << n << " ";
        if (n % 2 == 0) {
            n /= 2;
        } else {
            n = 3 * n + 1;
        }
    }
    std::cout << 1 << std::endl;
    return 0;
}
```
How to design a while loop

• Suppose you are attempting to implement an algorithm where you repeated apply a number of steps
  – How do you make the transition from manual to programmatic?

• Recommendation:
  – Do the algorithm on paper—in full
  – Examine the steps you took, and determine:
    • What steps were repeated?
    • What condition caused you to stop repeating the steps?
    • What local variables could you use?

The greatest-common divisor

• From secondary school, you saw that the algorithm for calculating the greatest common denominator (gcd)
  – You are asked to find the gcd of 8008 and 8085
  – You first note that 8085 > 8008
  – Next, you find that 8085 ÷ 8008 equals 1 with a remainder of 77
  – Next, you find that 8008 ÷ 77 equals 104 with a remainder of 0
  – From this, you are told that the gcd is 77

The greatest-common divisor

• Let's try again:
  – You are asked to find the gcd of 1583890 and 85800
  – You first note that 1583890 > 85800
  – Next, you find that 1583890 ÷ 85800 has a remainder of 39490
  – Next, you find that 85800 ÷ 39490 has a remainder of 6820
  – Next, you find that 39490 ÷ 6820 has a remainder of 5390
  – Next, you find that 6820 ÷ 5390 has a remainder of 1430
  – Next, you find that 5390 ÷ 1430 has a remainder of 1100
  – Next, you find that 1430 ÷ 1100 has a remainder of 330
  – Next, you find that 1100 ÷ 330 has a remainder of 110
  – From this, you are told that the gcd is 110

while ( m%n != 0 ) {
    int rem(m%n);
    m = n;
    n = rem;
}
std::cout << "The gcd is " << n << std::endl;
Thus, here is our program:

```cpp
int main()
{
    int m; int n;
    std::cout << "Enter the first integer: ";
    std::cin >> m;
    if (m < 0)
    {
        m = -m;
    }
    std::cout << "Enter a second integer: ";
    std::cin >> n;
    if (n < 0)
    {
        n = -n;
    }
    // Make sure m >= n
    if (m < n)
    {
        int tmp(m);
        m = n;
        n = tmp;
    }
    // Perform a slightly different gcd algorithm
    int rem(mn);
    while (rem != 0)
    {
        n = m;
        n = rem;
        rem = m\(\mod n\);
    }
    std::cout << "The gcd is ", n, std::endl;
    return 0;
}
```

Testing:

- Testing two prime numbers: the gcd should be 1
  - Enter the first integer: 157
  - Enter a second integer: 521
  
- Testing a multiple of a number: gcd should be smaller
  - Enter the first integer: 55241
  - Enter a second integer: 48609033
  
- Testing two relatively prime composites: gcd should be 1
  - Enter the first integer: 43010
  - Enter a second integer: 150423
  
- Testing two highly composite numbers: gcd should be 2310
  - Enter the first integer: 48510
  - Enter a second integer: 254100
  
The greatest-common divisor
The greatest-common divisor

- Testing with negative numbers:
  Enter the first integer: -157
  Enter a second integer: 521
  The gcd is 1
  Enter the first integer: 157
  Enter a second integer: -521
  The gcd is 1
  Enter the first integer: -157
  Enter a second integer: -521
  The gcd is 1

- Testing with zero: the gcd should be the other number
  Enter the first integer: 0
  Enter a second integer: 521
  Floating point exception (core dumped)
  – Issue, just like dividing by zero causes a program to terminate so does calculating m%0

The greatest-common divisor

- Thus, after we enter the numbers, we should check before we run the algorithm:
  // Make sure m >= n
  if (m < n) {
    int tmp = m;
    n = m;
    m = tmp;
  }
  if (n == 0) {
    std::cout << "The gcd is " << m << std::endl;
    return 0;
  }
  // Perform our gcd algorithm
  // ...

Every for loop can be written as a while loop

- The following two are essentially identical:
  int sum = 0;
  for (int k = 0; k < n; ++k) {
    sum += k;
  }
  int sum = 0;
  int k = 0;
  while (k < n) {
    sum += k;
    ++k;
  }
Infinite loop?

• Question:
  – What do you do if you accidentally execute a program that has an infinite loop?

• Solution:
  – In Eclipse, there is a stop button that becomes active when a program is executing
  – Other IDEs will have similar features
  – At the console, press Ctrl-C

Summary

• Following this lesson, you now
  – Understand how to implement while loops in C++
  – Seen how to implement various algorithms requiring looping statements:
    • Playing guessing games
    • Finding all prime factors
    • The Collatz conjecture
    • The factorial function
  – Understand how to convert a description of an algorithm to one that you can program
    • The example we used was the greatest-common divisor
  – Understand that all for loops can be written as while loops
  – Know how to terminate a program in an infinite loop

References

  https://en.wikipedia.org/wiki/While_loop

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

[3] tutorialspoint
  https://www.tutorialspoint.com/cplusplus/cpp_while_loop.htm

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.