Loops

CS A150 - C++ Programming 1

Loops

- 3 Types of loops in C++
  - **while**
    - *Most* flexible
    - No "restrictions"
  - **do-while**
    - *Least* flexible
    - Always executes loop body at least once
  - **for**
    - Natural "counting" loop

SYNTAX: `while` LOOP

Syntax for `while` and `do-while` Statements

A `while` STATEMENT WITH A SINGLE STATEMENT BODY

```
while (Boolean_Expression)
    Statement
```

A `while` STATEMENT WITH A MULTISTATEMENT BODY

```
while (Boolean_Expression)
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
}
```

EXAMPLE: `while` LOOP

- Consider:

```
int count = 1;              // Declare and initialize
while (count < 4)           // Loop Condition
{
        cout << "Hi ";      // Loop Body
        ++count;            // Update expression
}
```

- How many times does the loop body execute?

EXAMPLE: `while` LOOP

- Consider:

```
int count = 1;              // Declare and initiaze
while (count < 4)           // Loop Condition
{
        cout << "Hi ";      // Loop Body
        ++count;            // Update expression
}
```

- How many times does the loop body execute?

count = 1

count = 2

count = 3 → a total of 3 times

Book Example

- Display 2.4 – Example of a while Statement

SYNTAX: `do-while` LOOP

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

```
do
    Statement
while (Boolean_Expression);
```

A do-while STATEMENT WITH A MULTISTATEMENT BODY

```
do
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
} while (Boolean_Expression);
```

Do not forget the final semicolon.

EXAMPLE: do-while LOOP

- Consider:

```
int count = 0;              // Declare and initialize
do
{
        cout << "Hi ";      // Loop Body
        ++count;            // Update expression
} while (count < 0);        // Loop Condition
```

- How many times does the loop body execute?

EXAMPLE: `do-while` LOOP

- Consider:

```
int count = 0;              // declare and initialize
do
{
        cout << "Hi ";      // loop body
        ++count;            // update expression
} while (count < 0);        // loop Condition
```

- How many times does the loop body execute?
  - One time
- *Do-while* loops always execute body *at least* once

Book Example

- Display 2.5 – Example of a do-while Statement

DIFFERENCES: `while` VS. `do-while`

- Very similar, but…
  - One important difference
    - Issue is "WHEN" boolean expression is checked

      while → checks BEFORE body is executed

      do-while → checked AFTER body is executed
- Other than this, they are essentially identical!
- *while* is more common, due to its ultimate "flexibility."

Increment and Decrement Operators

- Although some programmers use increment and decrement operators in expressions, it is recommended you avoid it for better readability.

SYNTAX: `for` LOOP

```
for (initialization_statement; condition; update_expression)
        body_Statement
```

Example:

```
for (int i = 0; i <= 10; ++i)
{
    cout << i << " ";
}
```

- How many times does the loop execute?

SYNTAX: `for` LOOP

```
for (initialization_statement; condition; update_expression)
        body_Statement
```

Example:

```
for (int i = 0; i <= 10; ++i)
{
    cout << i << " ";
}
```

- How many times does the loop execute?
  - 11 times

For Loop: Incrementing

- A ***for*** loop can **increment *i*** more than just one unit at a time:

```
for (int i = 0; i < 10; i+=2)
{
    // statements…
}
```

- How many times will the for loop be executed?

For Loop: Incrementing

- A ***for*** loop can **increment *i*** more than just one unit at a time

```
for (int i = 0; i < 10; i+=2)
{
    // statements…
}
```

- How many times will the for loop be executed?
  - i = 0, 2, 4, 6, 8 → 5 times

Common Errors

- Watch the *misplaced semicolon* (;)
  - Example:

```
int num = 1;
string response = “”;

while (num != 0); ←
{
      cout << "Enter value: ”;
      cin >> response;
}
```

  - Notice the ";" after the while condition
  - Result here: INFINITE LOOP

Common Errors (cont.)

- Do **not** use **!=** to test the end of a numeric range.
  - Example:

```
for (int i = 0; i != num; ++i)
```

    - What if ***num*** is negative?
      - If *num* is -2, you will get an **infinite loop**

- Always use the ***strongest*** test:

```
for (int i = 0; i < num; ++i)
```

- **IMPORTANT:** **Never** test **floating-type** numbers for equality.

Infinite Loops

- **Loop condition** must evaluate to **false** at some iteration.
  - If not → **infinite loop**
  - Example:

```
while (1)
{
    cout << "Hello ";
}
```

  - A perfectly legal C++ loop → always infinite!

Processing Inputs

- When processing a sequence of input values, you need some way of knowing when you have reached the end.
- A special value used to **signal termination** is called a ***sentinel value*.**
- Do not use arbitrary values → use valid values such as 999, 9999, etc.

THE `break` AND `continue` STATEMENTS

- Flow of Control
  - Recall how loops provide "graceful" and clear flow of control in and out.
  - In **RARE** instances, you can alter natural flow.
- **`break;`**
  - Forces loop to exit immediately.
- **`continue;`**
  - Skips rest of loop body.
- These statements violate natural flow.
  - Only used when *absolutely* necessary!

- **Bottom line:** We will **<u>not</u>** use them.

Nested Loops

- **Recall:** ANY valid C++ statements can be inside body of loop.
- This includes additional loop statements
  - Called "nested loops"
- Requires careful *indenting*:

```
for (int outer = 0; outer < 5; ++outer)
     for (int inner = 7; inner > 2; --inner)
          cout << outer << inner;
```

  - Notice no { } since each body is one statement.
  - Good style dictates we use { } anyway.

Tip

- *Avoid* declaring the looping variable *outside* the for loop:

```
int i;   ← NO
for (i = 0; i < 5; ++i)
    //some statement
```

- Declare the variable inside the loop so that the **scope** of the variable is ***local***:

```
for (int i = 0; i < 5; ++i)
    //some statement
```

Introduction to File Input

- We can use **cin** to read from a file in a manner very similar to reading from the keyboard.
  - Only an introduction is given here, more details are in chapter 12.
  - Just enough so you can read from text files and process larger amounts of data that would be too much work to type in.

Opening a Text File

- Add at the top include

```
#include <fstream>
using namespace std;
```

- You can then declare an input stream just as you would declare any other variable.

```
ifstream inputStream;
```

- Next you must connect the **`inputStream`** variable to a text file on the disk.

```
inputStream.open("filename.txt");
```

  - The **`"filename.txt"`** is the pathname to a text file or a file in the current directory.

Reading from a Text File

- Use

```
inputStream >> var;
```

- The result is the same as using `cin >> var` except the input is coming from the text file and not the keyboard.

- When done with the file close it with

```
inputStream.close();
```

  - **IMPORTANT:** Make sure you **close the file** to avoid corrupting the file.

Book Example

- Display 2.11 – Program to Read a Text File
- Display 2.12 – Using a Loop to Read a Text File

Questions?

(Loops)