Do-While Loops, Summary of Control Flow Structures, and Random Numbers

Thomas Merkh
tmerkh@math.ucla.edu

Section 2, Week 4

April 23 2020
Overview

1 The Do-While Loop

2 Summary and comparison of Control Flow Structures

3 Random Number Generation

4 Next Time
Last Time

Last time we discussed nested loops and truth tables. As a refresher:

- Nesting for loops is useful when we need to iterate over nested objects (files inside folders, space and time, or just a table of data)
- Algorithmic efficiency is a way to discuss how long a certain code will theoretically take. This becomes important when nesting loops, due to the curse of dimensionality
- We also covered truth tables, Boolean operations, and DeMorgan’s laws
Last Time

Last time we discussed nested loops and truth tables. As a refresher:

- Nesting for loops is useful when we need to iterate over nested objects (files inside folders, space and time, or just a table of data)
- Algorithmic efficiency is a way to discuss how long a certain code will theoretically take. This becomes important when nesting loops, due to the curse of dimensionality
- We also covered truth tables, Boolean operations, and DeMorgan’s laws

Today, we will cover the final type of loop, summarize all of the control structures, and touch on random number generation in c++.
The Do-While Loop

Summary and comparison of Control Flow Structures

Random Number Generation

Next Time
Consider the problem where we would like to prompt a user for a positive number. With what we’ve learned so far, we would probably do this by:

```cpp
#include <iostream>
using namespace std;

int main() {
    double N = 0;
    cout << 'Enter a positive number:' << endl;
    cin >> N;
    while (N <= 0) {
        cout << 'Enter a positive number:' << endl;
        cin >> N;
    }
    return 0;
}
```

This is fine, but we need to paste the entire inner-loop code before the loop itself. How can we get around doing this?
Motivation for Another Loop

Consider the problem where we would like to prompt a user for a positive number. With what we’ve learned so far, we would probably do this by:

```cpp
#include <iostream>
using namespace std;

int main() {
    double N = 0;
    cout << "Enter a positive number:"
         << endl;
    cin >> N;
    while (N <= 0) {
        cout << "Enter a positive number:"
             << endl;
        cin >> N;
    }
    return 0;
}
```

This is fine, but we need to paste the entire inner-loop code before the loop itself. How can we get around doing this?
The Do-While loop exists just for this reason. Consider the following code:

```cpp
#include <iostream>
using namespace std;

int main() {
    double N = 0;
    do{
        cout << "Enter a positive number: " << endl;
        cin >> N;
    } while (N <= 0);
    return 0;
}
```

This accomplishes the same task, without needing to copy the inner-loop code prior to the loop itself.
The Do-While loop exists just for this reason. Consider the following code:

```cpp
#include <iostream>
using namespace std;

int main() {
    double N = 0;
    do{
        cout << 'Enter a positive number:' << endl;
        cin >> N;
    } while (N <= 0);
    return 0;
}
```

This accomplishes the same task, without needing to copy the inner-loop code prior to the loop itself.
The syntax for a do-while loop is

```c
int main() {
    do {
        // codes;
    } while (testExpression);
    return 0;
}
```

This loop differs from the while loop in that the inner-loop code executes before the condition is checked, and therefore will execute at least once. Sometimes this is referred to as an exit-controlled loop.
A Do-While Example

Here is another example, which takes the sum of the numbers the user provides:

```cpp
#include <iostream>
using namespace std;
int main()
{
    float number, sum = 0.0;

    do {
        cout << "Enter a number: ";
        cin >> number;
        sum += number;
    } while (number != 0.0);

    cout << "Total sum = " << sum;

    return 0;
}
```

A comparison between all of the loops will be presented in the following slides, but this is the end of learning control flow structures.
A Do-While Example

Here is another example, which takes the sum of the numbers the user provides:

```cpp
#include <iostream>
using namespace std;
int main()
{
    float number, sum = 0.0;

    do {
        cout << "Enter a number: ";
        cin >> number;
        sum += number;
    } while (number != 0.0);

    cout << "Total sum = " << sum;

    return 0;
}
```

A comparison between all of the loops will be presented in the following slides, but this is the end of learning control flow structures.
1. The Do-While Loop

2. Summary and comparison of Control Flow Structures

3. Random Number Generation

4. Next Time
Summary of Loops

There are three types of loops in C++:

- The `for` loop, an entry-controlled loop which performs an initial action, checks a test-condition, and performs an (increment) action after each iteration. This loop can equivalently be written as a `while` loop.

- The `while` loop, an entry-controlled loop which continues to iterate until the test-condition is false. This loop may never begin if the test-condition is initially false.

- The `do-while` loop, an exit-controlled loop which checks the test-condition after an iteration has ended. This loop always performs at least 1 iterate.

Each loop has its place in well coded C++ scripts, so it is worth knowing them all.
There are two main conditional statements in C++,

- The `if, else if, else` statement, first checks whether or not the `if` condition is true, if so, it performs the code block following and ignores the `else if, else` statements. If the condition is false, it moves on to the arbitrary number of `else if` statements in order, and if none of those are true, it defaults to the `else` code block.

- The `switch` statement, typically used for coding clarity and speed when checking many potential cases, this conditional statement performs the same function as the `if, else if, else` statement, where it tests an integer value rather than a boolean.
1 The Do-While Loop

2 Summary and comparison of Control Flow Structures

3 Random Number Generation

4 Next Time
Random Number Generation

Often, we will write algorithms which depend on pseudo-random numbers, i.e. numbers which statistically appear random but are generated deterministically by a computer. Generating random numbers requires:

- Including `<cstdlib>`, the C++ compatible version of the C standard library
- Providing the random number generator (RNG) a seed, or a number to initialize the RNG’s algorithm
  - Tip: Providing the same seed number results in the same sequence of generated numbers
  - Often, users will provide the current time as a seed, meaning the code will change each time that it is run
- Calling the `rand()` function, which returns an int in $[0, RAND_MAX] \supseteq [0, 32767]$

Side note: Some people have actually used weather data as a "natural way" to generate random numbers in a undeterministic way. Statistically, this has little to no benefit, but is still interesting.
Random Number Generation

Often, we will write algorithms which depend on pseudo-random numbers, i.e. numbers which statistically appear random but are generated deterministically by a computer. Generating random numbers requires:

- Including `<cstlib>`, the C++ compatible version of the C standard library
- Providing the random number generator (RNG) a seed, or a number to initialize the RNG’s algorithm
  - Tip: Providing the same seed number results in the same sequence of generated numbers
  - Often, users will provide the current time as a seed, meaning the code will change each time that it is run
- Calling the `rand()` function, which returns an int in $[0, \text{RAND\_MAX}] \supseteq [0, 32767]$

Side note: Some people have actually used weather data as a “natural way” to generate random numbers in an undeterministic way. Statistically, this has little to no benefit, but is still interesting.
```cpp
#include <cstdlib>  // srand, rand */
#include <iostream>

int main ()
{
    int iSecret, iGuess;
    srand(7);  // Optional – set ‘seed’ */
    iSecret = rand() % 10 + 1;  // generate # in 1 and 10 */
    do {
        cout << "'Guess the number between 1 an 10: '" << endl;
        cin >> iGuess;
        if (iSecret < iGuess)
            cout << "'The secret number is lower'" << endl;
        else if (iSecret > iGuess)
            cout << "'The secret number is higher'" << endl;
    }
    while (iSecret != iGuess);
    cout << "'It's about time!'" << endl;
    return 0;
}
```

Thomas Merkh  tmerkh@math.ucla.edu  April 23 2020  14 / 17
Random Number Generation

RNGs show up often, especially in science and technology. Applications include

- Monte Carlo simulations
- Electronic Games
- Cryptography
- Machine learning, optimization, and artificial intelligence
- Mathematics - Integrating high dimensional integrals is actually faster and more accurate when utilizing RNGs
- Modelling stochastic anything - financial markets, weather, physics, traffic flow, spread of illness
1. The Do-While Loop

2. Summary and comparison of Control Flow Structures

3. Random Number Generation

4. Next Time
Next week, we will be discussing functions! Perhaps one of the most essential components of any computer program.