Control Flow, Loops, and Switch Statements

Thomas Merkh
tmerkh55@gmail.com

Section 2, Week 3

April 16, 2020
Overview

1. Control Flow

2. Details about Loops

3. Next Time
Last Time

Last time we discussed classes and control flow. As a refresher:

- **Classes** are fundamental components of most C++ programs. They can be used to define new data types, and the operations defined for those types.

- **if, else if, else** statements are the first instance of control flow we’ve seen, where we can have a program “change direction” if some condition is true.

- Such **true or false** conditions commonly involve relational operators such as <, >, ==, !=, >= and so on.
Last Time

Last time we discussed classes and control flow. As a refresher:

- Classes are fundamental components of most C++ programs. They can be used to define new data types, and the operations defined for those types.

- if, else if, else statements are the first instance of control flow we’ve seen, where we can have a program “change direction” if some condition is true.

- Such true or false conditions commonly involve relational operators such as <, >, ==, !=, >= and so on.

If we don’t want to spend our days copying and pasting the same code over and over, we are going to need to learn about functions and loops.
1 Control Flow

2 Details about Loops

3 Next Time
Consider the code:

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

int main() {
    for (int i = 0; i < 4; i++)
    {
        cout << i * i << endl;
    }
    return 0;
}
```

What do we think the console will output?
Control Flow: For Loops

Consider the code

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

int main() {
    for (int i = 0; i < 4; i++)
    {
        cout << i * i << endl;
    }
    return 0;
}
```

What do we think the console will output? You were correct if you answered:

0
1
4
9
The For Loop

Lets break this code down

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

int main() {
    for (int i = 0; i < 4; i++)
    {
        cout << i * i << endl;
    }
    return 0;
}
```

- The token `for` indicates that it is a loop, or something to repeat a number of times
- The condition `(int i = 0; i < 4; i++)` can be understood as
  - First initializing a temporary variable `i = 0`
  - Performing the content inside the loop if `i < 4`
  - Last, when the content inside the loop is performed, increment `i` by 1 using `i++`
We haven’t explicitly covered everything in these notes. For example, we haven’t discussed

- The increment operator, $i++$, which means "add 1 to $i"$\(^1\)

- The `{ }` tokens define different scopes, where in simple programs, variables initialized in an inner scope will not exist once the scope is exited. In the `for` loop, the variable $i$ will not exist after the loop

- The `endl` statement is technically different from the `\n` statement you may have seen in formatting strings. Don’t worry about this until detailed advanced C++ classes

\(^1\)There is a technical difference between $i++$ and $++i$, but don't worry about this yet
What do you think the following code does?

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

int main() {
    string message = "Hello Earthing, The sun says hello";
    for (int i = 0; i < message.length(); i++)
    {
        if (isupper(message[i])) {
            cout << message[i] << endl;
        }
    }
    return 0;
}
```

This code iterates over each letter in the message string, checks if the given letter is upper case. If so, it prints it out to the console. The output sans newlines would be `H E T`. 
What do you think the following code does?

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

int main() {
    string message = "Hello Earthing, The sun says hello";
    for (int i = 0; i < message.length(); i++)
    {
        if (isupper(message[i])) {
            cout << message[i] << endl;
        }
    }
    return 0;
}
```

This code iterates over each letter in the message string, checks if the given letter is upper case. If so, it prints it out to the console. The output sans newlines would be H E T.
For loops are essential for real codes. A simple example from some current COVID19 research trying to understand how information spreads via Twitter is

- First, obtain a (huge) data set full of people’s public tweets
- Then, loop over each Tweet, and count the number of times the words COVID, covid, COVID-19, etc. occur
- Plot the number of tweets containing COVID versus day of the year since December

Given that we need to sift through hundreds of millions of tweets, we wouldn’t want to do this without loops!
While loops are our second type of loop. These loops take the structure

```java
while (some condition is true)
/* do something */
```

- If these seem like they achieve the same purpose as for loops, then you’d be correct! All for loops can be written as while loops, and vice versa. This will be shown in a later example.

- While loops tend to be more “dangerous” than for loops, since if the test condition is always true (maybe due to a bug in the code), the loop will go on forever.
While Loops

Consider the example

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

int main() {
    string message = "Hello Earthing, The sun says hello";
    int i = 0;
    while (i < message.length())
    {
        if (isupper(message[i])) {
            cout << message[i] << endl;
        }
        i++;
    }
    return 0;
}
```

This does the same exact thing as we have already seen from the for loop, this prints out H E T.
While Loops

What happens if the i++ statement was forgotten at the end?

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

int main() {
    string message = "Hello Earthing, The sun says hello’’;
    int i = 0;
    while (i < message.length()) {
        if (isupper(message[i])) {
            cout << message[i] << endl;
        }
        i++;
    }
    return 0;
}
```

The answer is that i would equal 0 forever, and the console would print out H H H H H H H H H H H H H H H H H H H H H H H a million times before you could cancel the program.

Thomas Merkh tmerkh55@gmail.com
April 16, 2020 12 / 22
While Loops

What happens if the i++ statement was forgotten at the end?

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

int main() {
    string message = "Hello Earthing, The sun says hello";
    int i = 0;
    while (i < message.length()) {
        if (isupper(message[i])) {
            cout << message[i] << endl;
        }
        i++;
    }
    return 0;
}
```

The answer is that i would equal 0 forever, and the console would print out H H H H H H H H H H H H H H H H H H H H H H H H H H a million times before you could cancel the program.
While Loops

The previous example is called an *infinite loop*. Something to think about - can for loops by themselves ever be infinite loops?

Why would anyone use while loops over for loops?

- Technically, for any while loop, there is an equivalent for loop, so it comes down to preference.

- While loops tend to be more natural for many tasks, whereas for loops tend to be more natural for others.

- At minimum, they both exist, and you will encounter code which utilizes both. To understand others’ codes, understanding for and while loops is necessary.
The switch statement sometimes comes in handy, consider:

```cpp
int main () {
    char grade = 'D';

    switch (grade) {
        case 'A':
            cout << "Excellent!" << endl;
            break;
        case 'B':
            cout << "Great Job" << endl;
        case 'C':
            cout << "Well done" << endl;
            break;
        case 'D':
            cout << "You passed" << endl;
            break;
        case 'F':
            cout << "Better try again" << endl;
            break;
        default:
            cout << "Invalid grade" << endl;
    }
    cout << "Your grade is " << grade << endl;
}
```
The switch statement is nice when there are many cases. This isn’t immediately clear from the programmer’s vantage point, but the switch statement is actually faster to run than using many if-else statements. This is a compiler-runtime trick, and you do not need to know the details.

Note that any switch statement can be converted to a large if, else if, else statement, and vice versa.

Last, sometimes they are preferred over nested if-else statements because changing nested statements can be more complex for the programmer.
1 Control Flow

2 Details about Loops

3 Next Time
Early Stopping

What is this code doing? How many iterations does the loop do?

```cpp
#include <iostream>

void findElement(int arr[], int size, int key)
{
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            cout << 'Element found at position ' << i + 1;
            break;
        }
    }
}

int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = 6;
    int key = 3;
    findElement(arr, n, key);
    return 0;
}
```

Answer: This code searches through the array of ints until it reaches 3, where it quits. This takes 3 iterations.
What is this code doing? How many iterations does the loop do?

```cpp
#include <iostream>

void findElement(int arr[], int size, int key)
{
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            cout << 'Element found at position ' << i + 1;
            break;
        }
    }
}

int main() {
    int arr[] = { 1, 2, 3, 4, 5, 6 };
    int n = 6;
    int key = 3;
    findElement(arr, n, key);
    return 0;
}
```

Answer: This code searches through the array of ints until it reaches 3, where it quits. This takes 3 iterations.
This code utilizes `break;`. This statement breaks the current loop at the moment that it is executed. We have not learned about nested loops yet, so we will revisit this later.

- Until then, if we write a loop with an objective that gets satisfied early on in the loop, we can utilize the break statement to prevent unnecessary computation
Infinite loops were introduced earlier. They happen. What are the common pitfalls and mistakes which lead to them?

- While loops that forget to increment a counter
- For loops which always have their continue statement true, like `for(int i = 1; i > 0; i++)`
- Loops which depend on a break statement, which never gets executed

Using Visual Studio, when an infinite loop begins, if the IDE doesn’t catch it then just go to Menu → Cancel Build.
Extra: Recursion

We probably will not be learning about recursion this quarter, but here is an example anyway:

```c
double factorial(double k) {
    if (k == 1 || k == 0) { return 1; }
    else { return k * factorial(k - 1); }
}
```

There are a couple things going on here:

- The function `factorial` is calling itself, unless `k = 0` or `k = 1`.
- This uses control flow to decide whether or not it should continue to call itself (similar to a while loop, though no loop is present). If an infinite recursion is attempted, you’ll get a stack error.
- Unsurprisingly, this function computes `k!`! Think about how this works, if you’re interested.
1 Control Flow

2 Details about Loops

3 Next Time
Next time we will cover more types of loops and how to nest loops within one another. Have a great weekend.