Control structures

Basic C++

Mike Burrell

March 5, 2024

Textbook readings

Chapter 1 — 1.4

Chapter 4 — 4.3

Chapter 5 — 5.1, 5.2, 5.3, 5.4

Goals for this set of slides

- Understand how to break up problems using `if`, `else`, `while`, `do`, and `switch`
- Understand type inference in C++

More history

- One of the earliest divergences between C and C++ is over the use of booleans
- From 1970 to 1999, C did not have *any* booleans
- In contrast, C++ had booleans right from the 1980s, but it maintained compatibility with boolean-less C
- This is important because it changes how we think about conditions (e.g., `if` statements)

Booleans in C++

- The word `bool` is a keyword (reserved word) in C++
  - As are `true` and `false`
- It exists outside of the usual integer type hierarchy
  - Its representation is completely implementation-defined
  - Most commonly, it is represented as a single byte (`sizeof (bool)` is very often 1)
- However, it *is* an integer type, of sorts. . . .

Integers and booleans

- To maintain better compatibility with C (which historically didn't have a `bool` type), C++ treats `bool`s as integers

false	— is defined to be 0
true	— is defined to be 1

- An integer will be implicitly converted into a `boolean`
  - Any non-zero value will be interpreted to be `true`
- Arithmetic (-, +, --, ++, etc.) is possible on `bool`s, too, though discouraged

Idiomatic C++

```cpp
int num_factors(unsigned int x)
{
    if (!x) {
        return 0;
    }
    int c = 1;
    for (unsigned int i = 2; i < x; i++) {
        if (x % i == 0) {
            c++;
            x /= i;
        }
    }
    return c;
}
```

Note the use of `if (!x)`

Integers as booleans

- Many C++ programmers (especially those who also use C) will idiomatically use integers as if they were booleans and vice versa
    - Also with pointers, which we'll see before long
- The behaviour that 0=false and anything-other-than-0=true is well-defined and usually a safe thing to take advantage of
- Just make sure that your code is clear and understandable

Most structures are the same

- `if`, `while` both work the same in C++ as they do in Python
    - Except that the conditions can be integers instead of booleans
- Like in Python, an `else` is possible, and `else ifs` may be chained together indefinitely

Boolean operators are the same

- `<`, `>`, `<=`. `>=`, `==`, `!=`, `&&` (and), `||` (or), `? :`, `!` (not), etc.
- Just be aware of the fact that the result of a boolean expression could be turned into an integer at any moment
  - E.g., `int x = (y < z) * 10;`
  - If $y < z$, then x will be 10; otherwise it will be 0.

For loops

- Basic for loops (`for ( ; ; )`) are commonly used in C++
- For-each loops (*enhanced for loops* in Java, *range-based for loops* in C++) are different though!
    - For-each loops in C++ are considerably more flexible and complex
    - Even with arrays, C++ for-each loops offer a lot of flexibility
    - We will look at these when we discuss vectors

Switch statements

- C++ has a `switch` statement that can take the place of `else if` in some instances
- In C++, `switch` statements may *only* be used with integer constants
- “Integer constants” includes enumerations, which we’ll discuss later in the course

Conclusion of control flow

- Use basic `if`, `for`, `while`, etc., as you would in Python
- Be aware of the fact that integers and booleans are interchangeable
- false=0, true=1, 0=false, non-zero=true

Scope

- Scope of variables works similarly, but not exactly the same as in Python
  - Curly-braces demark the scope of a variable
  - Variables are deallocated when they fall out of scope
- C++ has globals (declared outside of any scope)
  - The `static` keyword can be used to turn a global variable into a variable accessible only within the current file

Conclusion

- Variables work generally as they do in Python or C
- Booleans and integers are often freely mixed together