Topics to Learn Later

Kate Gregory

www.gregcons.com/kateblog

C++ has a LOT of syntax

- **Lots of ways to do things**
  - Some are faster
  - Some are more convenient
  - Some are holdovers from C++98 or C
- **You don’t need to know all of it to write a program**
- **Most of it will make more sense when you’ve written some programs**
  - You’ll have a problem to solve that the syntax deals with
  - You’ll know how to try using the syntax
- **But – you might come across something in other material**

Debugging

- **Whatever compiler you use, there is a debugger for you**
- **Debugging is a vital skill for all developers**
- **Not just to find bugs**
    - Understand flow of control
    - Watch values change
    - See when compiler calls things for you
        - Eg constructor
- **Learning to use your debugger is the first step towards being a better developer**

Casting

- **C++ is a strongly typed language**

```
int i = 4.9;
```

- **Compiler warnings or unexpected runtime values can be caused by "mixing and matching" types**
- **Casting tells the compiler "I meant to do that"**
  - Suppresses the warning
- **Casting tells other developers "look what I'm doing here"**
  - Makes intent obvious
  - Can be a place to spot cause of strange runtime values

```
i = static_cast<int>(4.9);
```

- **There are other cast templates for more dramatic casting**
  - dynamic_cast<>
  - const_cast<>
  - reinterpret_cast<>

The const keyword

- **Promises the compiler that a variable's value won't change**
  - Prevents logic errors
  - Enables optimizations

```
const int amount = 90;
```

- **Promises that a member function won't change the value of any member variables**

```
string Transaction::Report() const
{
// …
}
```

- **Add to function declaration and definition**

The Standard Library

- So much more than just <iostream>, <string>, and <vector>
- Collections
- Algorithms (find, sort, …)
- Complex numbers, random numbers, regular expressions
- . . .

- Standards committee is hard at work adding more
- Looking for a library? Check the Standard Library first

Passing Parameters to Functions

- **By default, what goes to the function is a copy**

```
void foo(Transaction t);
//…
Transaction deposit(50, “Deposit”);
foo(deposit);
```

  - Changes inside foo() will be to the local variable, not to deposit
- **You can arrange for the function to take the parameter by reference**

```
void foo(Transaction& t);
```

  - Call it exactly the same way: `foo(deposit);`
  - Changes to deposit will “stick”
- **Even if you don’t want to change the parameter, you might pass by reference**
  - Old school developers did this to save the runtime “cost” of a copy operation

```
void foo(const Transaction& t);
```

  - It expresses your intent, and ensures you won’t accidentally change the parameter

Classes That Manage Resources

- **Member functions**
  - Open, read, and write a file
    - Keep a file handle in a member variable
  - Work with a database
    - Keep an open database connection in a member variable
  - . . .
- **How can you ensure the resource is properly managed?**
  - Don’t leave the file hanging open
- **Could write a function**
  - Close, Dispose, Cleanup, …
  - People forget to call
- **C++ has a destructor**
  - Guarantees that cleanup gets a chance to happen
  - Name is ~ and name of class – Eg ~Account()

Scope

```
{
    int i;
    Account a;
    Transaction T(50,"Deposit");
}
```

- **Constructor runs when object comes into scope:**
- **Destructor runs when object goes out of scope**
- **Most common case – flow of control reaches closing brace**
- **Member variables go out of scope when the instance they belong to does**

Things to Learn Elsewhere

- **Exceptions**
  - Alternative to returning error codes
  - Can make neater and faster code when done right
- **The free store**
  - Raw pointers
  - `std::shared_ptr` and `std::unique_ptr`
  - Memory management – and resource management in general
  - Learn from modern material only!
  - RAII, Rule of 3, Rule of 5
- **Lambdas**
  - A way to use a few lines of code as a parameter to a function, or something to store in a variable

Minor details

- Inheritance, virtual functions, polymorphism, multiple inheritance
- the enum keyword
- Boolean operators && and ||, shortcutting
- Interacting with the OS – eg calling a Windows API
- Bitwise operators & | ^ ! << >>
- The switch statement
- More punctuation you haven't seen yet
    - %
    - & * ->
    - ?
- Default parameters to functions
- Writing templates
- Writing your own operator overloads

Summary

- **You know enough C++ to write a real program**
- **You'll need to learn a lot more to write some kinds of applications**
  - Windows application (desktop)
  - Windows store application (Windows 8, 8.1, …)
  - Windows Phone application
  - Unix application
  - Web service
  - Service
- **Learn frameworks and libraries as a next step**
- **C++ has a lot of syntax**
  - Learn it when you need it
  - If something feels really hard, remember there is more C++ you can learn that might include an easier way to do it