

C versus C++ (Procedural Programming versus Object Oriented)

Original slides by Chris Wilcox,
Colorado State University

CS270 - Fall Semester 2015 1

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

C Versus C++

- ◆ Question: Aren't they really almost the same language? Isn't C++ just a superset of C? Answer: No, C++ is very different and immensely more powerful than C.
- ◆ Question: Can I take my C programs and turn them into C++ by adding objects around everything? Answer: Yes, but there's lots more to C++ than just object-oriented C.
- ◆ Question: Can I ignore C++ and move on to Java? Isn't that what everyone programs in now? Answer: Maybe, it depends on where you work and what you do.
- ◆ Question: Does the instructor of this course think that C++ is an amazing language. Answer: Of course, however I am aware that C++ has its own set of arcane problems.

CS270 - Fall Semester 2015 2

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

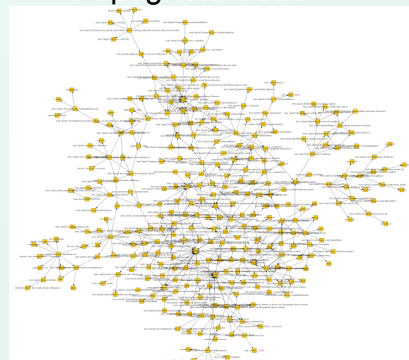
C Language

- ◆ What does the C language provide? Variables, constants, simple data types, compound data types, operators, control flow, pointers, functions.
- ◆ What is the structure of a C program? Really just an entry point, functions, and global data. Any function can call all other functions, anytime. Same is true for data access.
- ◆ What does the C language not provide? Objects, interfaces, encapsulation, inheritance, and standard mechanisms for threading, mutexes, semaphores, sockets, and timers. Also no containers and algorithms.
- ◆ Four 'C' dilemmas: 1) how to organize procedural code, 2) how to make programs portable, and 3) how to avoid writing defects, including pointer and memory management bugs!

CS270 - Fall Semester 2015 3

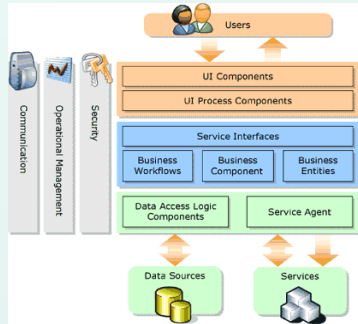
Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Spaghetti Code



CS270 - Fall Semester 2015 4

Layered Architecture



Procedural Programming

- ◆ Much effort has been spent trying to develop solutions that allow organization of procedural programs:
 - Define interfaces using application programming interfaces (APIs) to divide architectural layers.
 - Organize functions to maximize cohesion and minimize coupling between modules.
 - Create header files with related functions, essentially the equivalent of an object-oriented interface.
 - Avoid the use of global variables, group related data items into structures which can carefully managed.
- ◆ Does this solve the problem? Only if a disciplined approach is maintained, but this is rarely the case (in my experience).

Object Oriented Languages

- ◆ Group data and code into a single entity called an object, allowing encapsulation of complex internals.
- ◆ Key concept: separation of interface from implementation, allows an abstraction of functionality.
- ◆ Architects draw a block diagram of the entire system and identify and design interfaces.
- ◆ Public, protected, and private classification apply to data or methods within the object.
- ◆ Common practice: never allow external access to data objects, supply get and set methods instead.
- ◆ OO languages facilitate achieving low coupling which is enforced by the language itself.

Object Declaration

```
class Clookup {  
public:  
    void construct(vector<Table> vTables, U32 uLutSize);  
    void generate(string sPreamble);  
    void replace(string sReplace);  
private:  
    void analyze(Eanalysis eAnalysis, U32 uLut);  
    vector<Table> m_vTables;  
    vector<Variable> m_vVariables;  
    U32 m_uLutSize;  
};
```

Other C++ Features

- ◆ C++ standard template library (STL): containers and associated algorithms:
 - vector, list, deque, set, multiset, hash containers
 - find, count, sort, search, merge, count, bound
- ◆ C++ strings: a complete revision to the C character array and string functions, much more like Java:
 - string
- ◆ C++ iostream library, a complete revision of the C functions for input/output, but native to C++:
 - ios, istream, ostream,fstream, stringstream
- ◆ C++ memory management: a complete revision to the C malloc and free interface, but still explicit!
 - new, delete

C++ Missing Features

As compared to Java:

- ◆ Standard syntax for sockets
- ◆ Standard syntax for threading
- ◆ Standard syntax for synchronization (mutex, semaphore)
- ◆ Standard syntax for timing

Thus all of these remain operating system dependent!

C++ Strings Example

```
#include <string>
string s1 = "This is ";
string s2 = "a string";
string s3 = s1 + s2; // string concatenation
if (s1 == s2) // string comparison
int len = s3.length(); // string length
string s4 = s3.substr(0,5); // extract substring
int i = s3.find("is", 0); // find substring
s3.erase(3, 7); // erase substring
const char *oldstr = s3.c_str(); // C string
```

C++ Vector Example

```
#include <vector>
vector<int> vIntegers;
vector<float> vFloats;
vector<string> vStrings;
vIntegers.clear(); // clear the vector
vIntegers.push_back(1234); // add an entry
vIntegers.push_back(3456); // add an entry
vIntegers.size(); // return the size
vIntegers[0]; or vIntegers.at(0); // access element
vIntegers.insert(0, 2345); // insert element
```

C++ Streams Example

```
#include <fstream>
void Cfile::Read(const string &infile,
                vector<Cartesian> &vPoints) {
    ifstream inputFile(infile.c_str());
    if (inputFile) {
        Cartesian pt;
        while (inputFile >> pt.xCoord >> pt.yCoord
              >> pt.zCoord)
            vPoints.push_back(pt);
    }
    // inputFile.close();
}
```