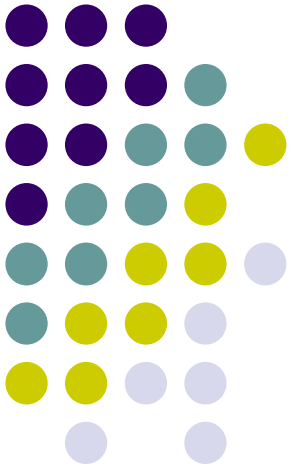
C++

January 8, 2015

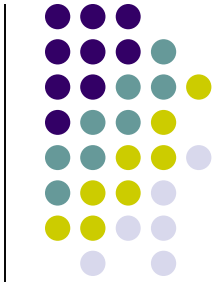
Mike Spertus

mike_spertus@symantec.com



This week's lecture

- Today is a survey of C++
  - Want to give some of the big picture without worrying too much about the technical details
  - Don't worry if some things are unclear or not covered in enough depth
  - We'll present language features systematically in the coming weeks

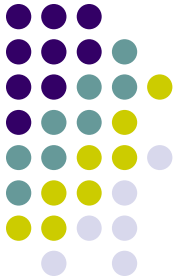


COURSE INFO

Resources

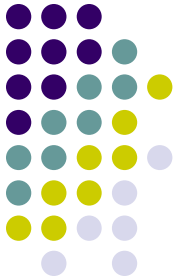
- Required:
  - C++14 Standard. Use near-final draft at http://open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3797.pdf
- Recommended:
  - http://isocpp.org/
  - http://www.open-std.org/jtc1/sc22/wg21/
  - http://herbsutter.com/ - Herb is the Convener of the standards committee
  - Koenig and Moo, Accelerated C++
  - Scott Meyers, Effective Modern C++ (C++ best practices)
  - Stroustrup: The C++ Programming Language: 4th edition
    - The inventor of C++' book on the language
    - Only use the 4th edition, which covers the (almost) current C++11 standard
  - Josuttis and Vandervoorde, C++ Templates: The Complete Guide
    - As you will see, much of the course will be about templates in one guise or other.
    - Unfortunately, they do not yet cover anything after C++98
    - New edition not due until 2016
  - Anthony Williams, C++ Concurrency in Action
    - C++11 multithreading. Our main topic for the last few weeks.
    - If you don't want to buy/read the whole book, Anthony's blog gives all you need in his multithreading in C++ series
      - http://www.justsoftwaresolutions.co.uk/threading/multithreading-in-c++0x-part-1-starting-threads.htm
- Whatever books/sites work for you

The most important rule



- If you are ever stuck or have questions or comments
- Be sure to contact me
  - mike@spertus.com
- Or your TA
  - Paul Bossi
    - Email and google chat: bossi.cpp@gmail.com
- Or your grader
  - Dan Mainka
    - dmainka@gmail.com

Homework and Lecture Notes



- Homework and lecture notes posted on chalk.uchicago.edu
  - Choose MPCS 51044 and then go to Lab/Lectures in the navigation pane
- Homework submission instructions are at https://phoenixforge.cs.uchicago.edu/svn/mpcs51044-win-15/svntut.html
  - Apparently, IT is still working on enabling that, but hopefully up soon
- Instructions and more are at http://mpcs51044-wiki.cs.uchicago.edu/
- Hopefully will have a Piazza group
- Due on the following Thursday before class
- Graded homework will be returned by the start of the following class
- ☹
  - Since I go over the answers in class, **no late homework will be accepted**
- ☺
  - If you submit by Sunday, you will receive a grade and comments back by Tuesday, so you can try submitting again

Grading

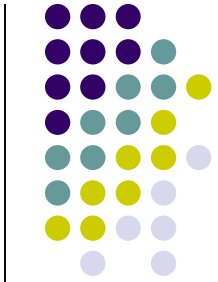
- 2/3 HW
  - Many extra credit opportunities
  - Extra credit can get your HW total for the quarter to 100% (but no higher) to cancel out any problems you miss
- 1/3 Final
  - The biggest part of the final is to do a code review of a willfully bad (but unfortunately not worse than some code you'll see in real-life)

C++ for programmers?

- Prior knowledge of C++ helpful but not required
- Expect you know how to program in some language
  - Not an introductory course on programming
- Gloss over features that are similar to those in Java, C, etc.: `if`, `for`, `?:`,...
  - OK if these are unfamiliar to you
    - Ask questions in class
    - Email or IM me or the Tas
    - Look up in the many recommended texts

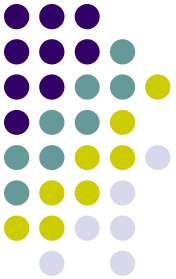
Most programmers learn C++ on the street—Bad idea

- It certainly didn't work for me
  - After over a decade of picking up C++ through general exposure and writing commercial C++ libraries, I thought I was an expert C++ programmer
  - Then I joined the C++ standards committee
  - Turns out I was a rank amateur
  - The real inventors of C++ routinely used many powerful C++ techniques and idioms that I had never heard of
    - If you don't know what RAII is, look it up immediately after this talk
- Their libraries were much more powerful, flexible, performant, and easy to use than any I had produced are even imagined



WHAT IS C++

What is C++: Hello World

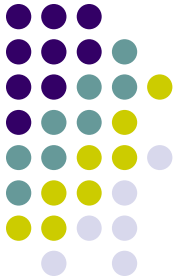


```
#include <iostream>

int
main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

A brief history of C++

- 1979 “C with classes” invented by Bjarne Stroustrup
- 1983 Renamed C++
- 1998 First standard.
- Boost libraries released
- 2003 A minor standard revision
  - Primarily fixed defects in the 1998 wording
- 2011 C++11 standard.
  - Called C++0X under development so “X” ended up being “b”! (We were overly ambitious)
  - Bjarne Stroustrup says it seems like a “Whole New Language”
  - This course will spend a lot of time on the new C++11 features
- 2014 C++14
  - The current standard
  - A smaller release with small “why didn’t we do that” features that make a big difference
  - We are now trying to release standards every three years

C++11



- Many new features
  - Threads
  - Memory model
  - Lambdas
  - Rvalue references
  - Initializer lists
  - “auto” variables
  - Many more
- http://www2.research.att.com/~bs/C++0xFAQ.html
- You’ll need a relatively new compiler to use these features
- A big part of the course
- See http://www.open-std.org/jtc1/sc22/wg21/ to understand how the designers of C++ think

C++14

- A smaller release with small “why didn’t we do that” features that make a big difference
  - More production-strength concurrency
    - Reader-writer locks are the most important
  - Polymorphic lambdas
    - Much bigger than it sounds
    - We’ll learn what and why later
  - Standard user-defined literals
    - If a function expects a time, you can pass it any of 1min, 60s, 60000ms, etc. to give it a minute

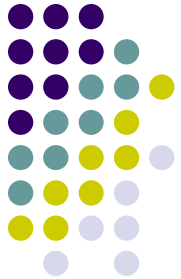
What C++ isn't

- C++ isn't a better C or a worse Java
- Don't be misled by superficial similarities to C and Java.
- Good C or Java code is not necessarily good C++ code
  - >90% of C++ programmers make this mistake
  - We will heavily emphasize best practices specific to C++
  - One of my goals today is to convince you that this is correct
    - Knowing how to program in Java, C, or other languages does not mean you know how to program in C++

What C++ is

- Bjarne Stroustrup, C++' inventor, moderated a discussion to come up with a "sound bite" description for C++
  - "C++ is a programmer's language"
  - "C++ is a flexible and expressive language"
  - "C++ is a multi-paradigm language"
  - "C++ is an industrial-strength toolset for the programming professional. "
  - "C++ is a high-performance general-purpose programming language"

C++ is a lightweight abstraction language



- Large, professional computer programs need to be written using abstraction and patterns to be maintainable and evolvable but at the same time they need to run efficiently
- Unlike almost any other language, C++ gives you the best of both world, allowing you to create powerful abstractions that are lightweight
- In other words, there is no performance penalty associated with using/creating abstractions

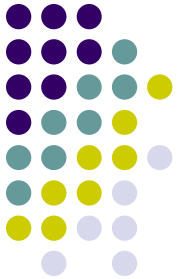
C++ is a standard

- The standard is your toolbox
- C++ is a large language. When questions arise (and they will), the standard is the authoritative answer
  - Standardized in 1998
  - Minor revision in 2003
  - Major update in 2011
  - Medium update in 2014
- Accurate in all significant ways but a “draft” to meet ISO requirements limiting free public access to the actual standard
- The current standard is C++14
  - http://open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3797.pdf is a freely available draft that is virtually identical to the official standard.
  - All modern compilers support C++98 very well
  - C++11 reasonably well
  - C++14 sporadically

C++ is not a standard

- Just as important as knowing what the C++ standard says, you need to know what it doesn't say
- Even simple code relying on "obvious" non-standardized behavior may be very fragile
- Relying on non-standard C++ behavior is necessary. E.g.,
  - Bits in an integer
  - DLLs
  - "There are no interesting standards-compliant program"
- However, it reduces portability and is fragile
- If you need to rely on non-standardized behavior (and you will), try to rely on "implementation-defined" rather than "undefined" behavior, so at least it is defined somewhere

Vectors



- You will want to learn at least a little about vectors for the HW. Basically, if what is in the code below makes sense by context, you're golden. Otherwise, ask me.

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

int main()
{
  vector<int> v = {1, 2}; // Init a vector of ints
  v.push_back(4); // Add an element to the end
  for(auto it = v.begin(); it != v.end(); it++) {
    cout << *it << ", ";
  } // Prints "1, 2, 4,"
  return 0;
}
```

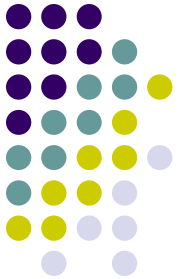
Defining functions

```
int square(int n)
{
  return n*n;
}

int main()
{
  cout << square(2);
  return 0;
}
```

Defining functions

```
int square(int n)
{
  return n*n;
}

double square(double n) // OK to have two functions
{                       // with same name (overloading)
  return n*n;           // as long as compiler can tell
}                       // which you mean by context
                        // We'll make this more precise
int main()              // in a few weeks
{
  cout << square(2) + square(3.1416);
  return 0;
}
```

Let’s look at some programs

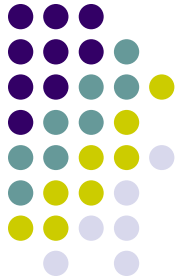


- Frame.cpp and Frame2.cpp from chalk

Generics

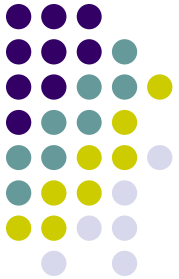
- The key to implementing lightweight abstractions is C++' powerful generic mechanism (also known as templates).
- Templates let you give a name to a not yet specified type
- We will spend more time on generics than any other topic
  - They are that important to modern C++
  - They are the most distinguishing feature of C++

Generic functions



```
template<typename T>
T square(T n)
{
  return n*n;
}

int main()
{
  return square(2) + square(3.1416);
}
```

Defining functions



```
template<typename T>
T square(T n)
{
  return n*n;
}

Circle square(Circle c)
{
  cout << "Cannot square the circle" << endl;
}

int main()
{
  return square(2) + square(3.1416);
}
```

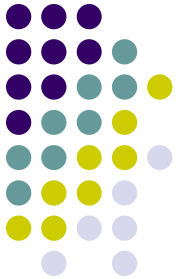
C++ is way too complicated for the average programmer. True or false?

- You hear this a lot
- On the surface, it appears to have a lot of truth
  - Most C++ teams that I interact with are “over their heads” understanding the subtleties of the C++ language features they are using
  - We’ll see some truly hair-raising C++ code later this lecture
- Nevertheless, I contend that this is a misconception based on misunderstanding how the language should be used

Two styles of programming

- C++ distinguishes between writing “library code” and “application code”
- Writing your application should be simple and clear
  - Should not need many scary and complex techniques
  - Most of your programmers should be writing application code in a clear and simple C++ dialect
  - Do need to be aware of Best Practices for writing applications
- Writing general purpose reusable template libraries can make use of all the power and functionality of C++
  - These libraries consolidate all of the complexity in one place, so the application programmer using these libraries is insulated from the complexity
  - To accomplish this, C++ has many advanced and powerful features that are essential to writing powerful and easy-to-use libraries but are best avoided in application code
  - Your senior programmers will write your libraries
    - They will find the advanced features of the language indispensible
    - And have the skills to use them

Example: vectors

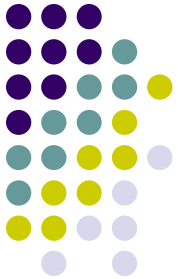


- Using a vector
  - Add up the elements in a vector
    - auto sum = accumulate(v.begin(), v.end(), 0.0);
  - Print the elements of a vector
    - Option 1: for(auto x : v) { cout << x << endl; }
    - Option 2: copy(v.begin(), v.end(), ostream_iterator<int>(cout, "\n“));
- Writing vector
  - Read warnings here: http://stackoverflow.com/questions/4304783/c-vector-source-code
  - Now look at the source http://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a01570_source.html
  - The C++ standard libraries can’t even use “new” to allocate memory or ordinary pointers to point at the allocated memory

Even application coding can be tricky at times

- This comes back to how most programmers pick up C++ “on the street.”
- Languages like Java are designed to minimize the risk from ignorant programmers
  - Don’t quote me on this ☺
- C++ assumes that as a professional programmer, you are willing to invest some time learning the language to get a more powerful tool

The goals of this class



- MPCS51044 will make you an expert C++ applications programmer
- MPCS51045 will make you an expert C++ library writer

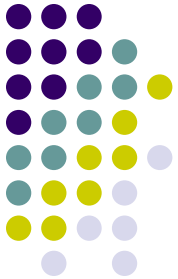
Example: Copy

- How do we copy?
- In C

```
memcpy(cp, dp, n);
```

- This works in C++, but what if we are copying to/from an
  - array of objects?
  - list?
  - stream?
  - …

std::copy



- C++ standard library provides a standard copy function

```
copy(sp, sp+16, destp);
```

- From vector to array

```
vector<char> v;
...
copy(v.begin(), v.end(), destp);
```

std::copy—(Cont)

- From array to vector
- If we just copy to the vector, we will write off the end of the vector.

```
copy(cp, cp+8, v.end());// Wrong!
copy(cp, cp+8, back_inserter(v));
```

- The point is that copy takes "iterators," and std::back_inserter creates an iterator that appends to the end of a vector

Iterators

- Roughly speaking, iterators in C++ generalize pointers to array elements in C
- Much more general
  - Can iterate an array
  - Can iterate a container
  - Can iterate a stream
  - Can be input or output
  - Can be sequential or random access
  - Can append
  - etc.

Can we copy to a stream?

- Sure, we just need to turn it into an iterator
- Printing a comma-delimited vector of doubles

```
vector<double> v;
...
copy(v.begin(), v.end(),
 ostream_iterator<double>(cout,", "));
```

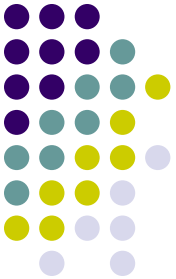
Comparing C and C++ copy

- C++ is better
  - The C++ copy is a general abstract encapsulation of the logical idea of copying anything to anything else, while the C memcpy command merely copies blocks of memory, which covers only a small portion of copying scenarios and requires the programmer to know about implementation details.
- C is better
  - Since memcpy knows you are simply copying blocks of memory, the compiler emits ultra-efficient hardware instructions to move memory.
- This is the abstraction vs. performance tradeoff mentioned above

What happens in real-life

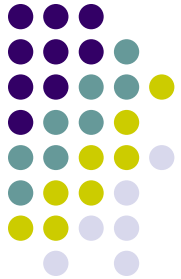
- Programmers traditionally use their knowledge of underlying types to either write a memcpy or a hand-coded object-based copy loop. Breaking encapsulation like this makes the code less robust and extensible:
  - What happens when a programmer working on one part of the code adds a smart pointer to a struct definition without being aware that some other part of the program memcpy's the struct
  - What happens when the array gets replaced by a vector?
  - What happens in generic code?

C++ to the rescue

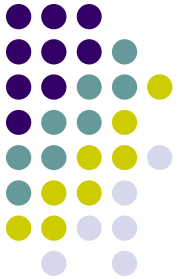


- Can we make it so the programmer can always call copy
  - The programmer just needs to know a single logical interface
- But the compiler generates code to move blocks of memory if that is safe?
  - The programmer knows that the compiler will understand what is being copied well enough to always generate the fastest possible machine code

Next week: We will show how this can be done

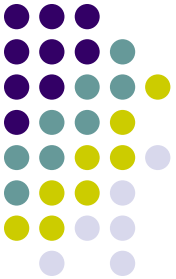


- We will use this to demonstrate how C++ can provide that most abstract, maintainable, and well-structured interface to the programmer without sacrificing performance
- In other words, “a lightweight abstraction language”
- Warning: This is to give you a glimpse of where we’re going. Don’t worry if you don’t understand it
  - You’re not expected to next week,
  - but it will be second nature by the end of the class ☺



HOMEWORK

HW 1.1

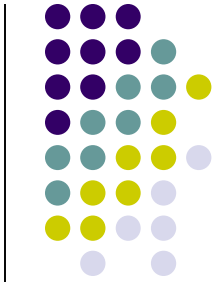


- The purpose of this exercise is to help you make sure you have a suitable C++ compiler installed and that you know how to build programs
- All referenced code is available on chalk
- Build, compile, and run the "Frame" programs on your compiler of choice.
- Send something to demonstrate that you've done this successfully (e.g., screenshots, any files you've written, including C++ files, makefiles, Visual Studio project files, a transcript of your shell session, etc.)
- Do not submit any executable files are large binaries! They aren't any good for the graders

HW 1.2

- Build, compile, and run the vector program from slide 20
  - You can download it from chalk
- This will make sure your compiler supports at least some C++11. Reasonably current versions of g++, Clang, and Visual Studio should be able to handle this easily

HW 1.3

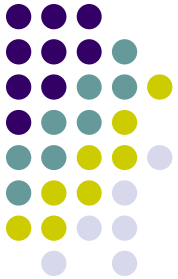


- Print out the first 8 rows of Pascal's triangle. This assignment is most easily completed by using a nested container, e.g., a vector<vector<int> >. Note in particular the need for the space in "> >", to avoid confusing the lexing stage of the compiler.
- If you've never seen C++ vectors, look at slide 17 for a simple example(or ask me or Paul)

HW 1.3 – Extra Credit

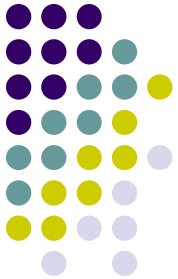
- For additional credit on the previous problem, it should be tastefully formatted. In particular
  - Use "brick-wall" formatting, in which the numbers of each row are presented interleaved with the numbers on the rows above and below.
  - The brick size should bethe maximum size of any integer in the triangle. For aesthetic as well as technical reasons, it is useful if the brick size is odd, so you may increase the size by one if necessary to make this true.
  - Each number should be centered on it's brick.

HW 1.4 (Extra Credit)



- Write a valid C program that is not a valid C++ program.
- Hint: There are ways to do this that don't require prior experience with C. Look for some simple "thinking-out-of-the-box" solutions.

HW 1.5 (Extra Credit)



- Why is C++ called C++ and not ++C?
- Note: If you are new to languages with the "++" operator, see
  - http://cplus.about.com/od/glossar1/g/preincdefn.htm
  - http://cplus.about.com/od/glossar1/g/preincdefn.htm