Introduction to C++: Part 4

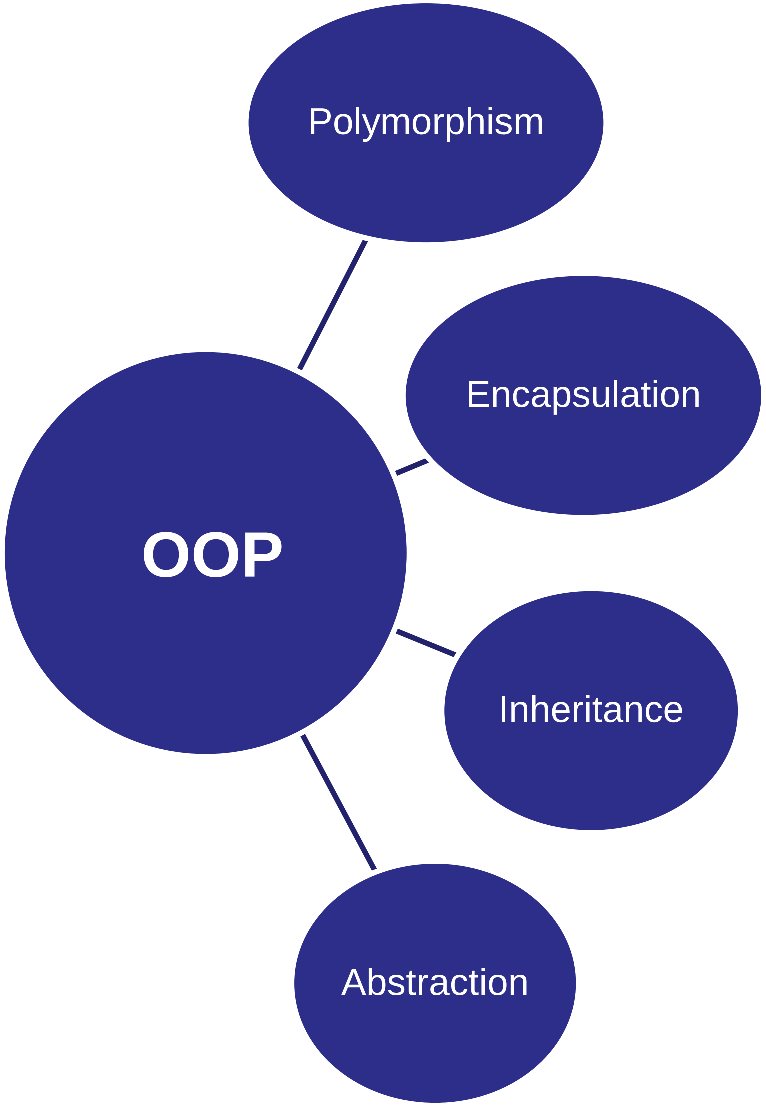
BOSTON UNIVERSITY

Tutorial Outline: Part 4

- Generics and templates
  - C++ template syntax
  - What happens during compilation
- Using generics: the C++ Standard Template Library (STL)
- STL Containers, Algorithms, and Iterators
- Coding recommendations for a C++ code
- Useful libraries
- Example: Speeding up Python
- Resources

The formal concepts in OOP

- Object-oriented programming (OOP):
  - Defines *classes* to represent data and logic in a program. Classes can contain members (data) and methods (internal functions).
  - Creates *instances* of classes, aka *objects*, and builds the programs out of their interactions.
- The core concepts in addition to classes and objects are:
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction



BOSTON UNIVERSITY

Polymorphism

- This has already been seen in the last tutorial:

```
void PrintArea(Shape &shape) {
    cout << "Area: " << shape.Area() << endl ;
}
```

- The different subclasses of Shape automatically call their own Area() method depending on their type.
  - …if virtual method calls are being used!
- This is called polymorphism in C++.
- There are two other kinds defined in computer science:
  - *ad hoc* polymorphism – function overloading in C++
  - *parametric* polymorphism – generics in C++

BOSTON UNIVERSITY

Function overloading

- Briefly: the same function can be implemented multiple times with different arguments.
- This allows for special cases to be handled, or specialized behavior for different types.
- Multiple constructors in a class are an example of function overloading.

```
float sum(float a, float b) {
    return a+b ;
}

int sum(int a, int b) {
    return a+b ;
}
```

Generics aka C++ Templates

- Generic code is code that works on multiple different data types but is only coded once.
- In C++ generic code is called a *template*.
- A C++ template is implemented entirely in a header file to define generic classes and functions.
- The actual code is generated **by the compiler** wherever the template is used in your code.
  - There is NO PENALTY when your code is running!
  - If you don't use the template code it doesn't get compiled at all.
- For the sake of time in this tutorial we will focus on *using* the C++ Standard Template Library and walk thru some templates with C::B.

Sample template function

- The template is started with the keyword *template* and is told it'll handle a type which is referred to as *T* in the code.
  - Templates can be created with multiple different types, not limited to just one.
  - You don't have to use *T*, any non-reserved word will do.
  - Methods inside a class can be template even if the class is not.
- When the compiler sees the call to the template function it will automatically generate a function that takes and returns float types.
  - If the compiler can figure it out you can sometimes skip the type declaration.

```
template <typename T>
T sum_template (T a, T b) {
    return a+b ;
}
// Then call the function:
float x=1.0 ;
float y=2.0 ;
float z=sum_template<float>(x,y) ;


z = sum_template (x,y) ;
```

BOSTON UNIVERSITY

Templates

- The only limit is that any type or class used with the example function *sum*<> has to support or implement the + operator

- If you use a template function or class and the type you want to use doesn't work with the generated code the compiler will tell you with an error message.
  - This may generate an ENORMOUS AMOUNT of error messages from the compiler.
  - If that happens, scroll back to the 1st error, that's usually the point in your code with the erroneous line creating a templated object.

- If you only have one type to worry about (e.g. only one type of image format), templates are unlikely to offer much (except longer compiles).
- Use them when needed by a library or when you find yourself repeating the same code for multiple types over and over.

BOSTON UNIVERSITY

A Template Class

- Open the Code::Blocks project:
  - Part 4/Overloads_and_Templates

- Let’s use the C::B debugger to walk through some function overloads, a template function, and a template class to see how the code is created by the compiler.

```
template <typename T>
class Sample
{
    public:
        Sample(T value) : m_stored_value(value) {}

        virtual ~Sample() {}   // <-- {} not ;
        // There's no .cpp file so all methods must
        // have a function body here.

        T sum_with_stored_value(T value) {
            return m_stored_value + value ;
        }
    protected:

    private:
        T m_stored_value ;
};
```

```
// Create an object of Sample cast to hold a specifc
type.
Sample<int> int_Sample(100) ;
cout << int_Sample.sum_with_stored_value(50) << endl ;
```

BOSTON UNIVERSITY

Template Class Inheritance

- C++ lets you define a base or super class using templates.
- A subclass can inherit as a template or as a specific type.
- Reference: C::B project Part 4/Template_Class_Inheritance

```
template<typename T>
class BaseClassTemplate
{
    public:
        BaseClassTemplate() {}
        virtual ~BaseClassTemplate() {}

        T m_base_value ;
    protected:

    private:
};
```

```
class Subclass1 :
     public BaseClassTemplate<int> {
public:
    Subclass1() {}
    virtual ~Subclass1() {}

    int m_some_new_val ;
};
```

```
template<typename T>
class Subclass2 :
     public BaseClassTemplate<T> {
public:
    Subclass2() {}
    virtual ~Subclass2() {}

    int m_some_new_val ;
};
```

```
template<typename T, typename Q>
class Subclass3 :
     public BaseClassTemplate<T> {
public:
    Subclass3() {}
    virtual ~Subclass3() {}

    Q m_some_new_val ;
};
```

The Standard Template Library

- The STL is a large collection of containers and algorithms that are part of C++.
  - It provides many of the basic algorithms and data structures used in computer science.
- As the name implies, it consists of generic code that you specialize as needed.
- When developing C++ code it is a good idea to use the STL when possible.
  - Well-vetted and tested.
  - Lots of resources available for help.
  - Programming is hard enough – why write extra code if you don’t have to?

Containers

- There are 16 types of containers in the STL:

Container	Description
array	1D list of elements.
vector	1D list of elements
deque	Double ended queue
forward_list	Linked list
list	Double-linked list
stack	Last-in, first-out list.
queue	First-in, first-out list.
priority_queue	1^{st} element is always the largest in the container

Container	Description
set	Unique collection in a specific order
multiset	Elements stored in a specific order, can have duplicates.
map	Key-value storage in a specific order
multimap	Like a map but values can have the same key.
unordered_set	Same as set, sans ordering
unordered_multiset	Same as multisetset, sans ordering
unordered_map	Same as map, sans ordering
unordered_multimap	Same as multimap, sans ordering

BOSTON UNIVERSITY

Algorithms

- There are 85 of these.
  - Example: find, count, replace, sort, is_sorted, max, min, binary_search, reverse
- Algorithms manipulate the data stored in containers but is not tied to STL containers
  - These can be applied to your own collections or containers of data
- Example:

```
vector<int> v(3);                  // Declare a vector of 3 elements.
v[0] = 7;
v[1] = 3;
v[2] = v[0] + v[1];                // v[0] == 7, v[1] == 3, v[2] == 10
reverse(v.begin(), v.end()) ;  // v[0] == 10, v[1] == 3, v[2] == 7
```

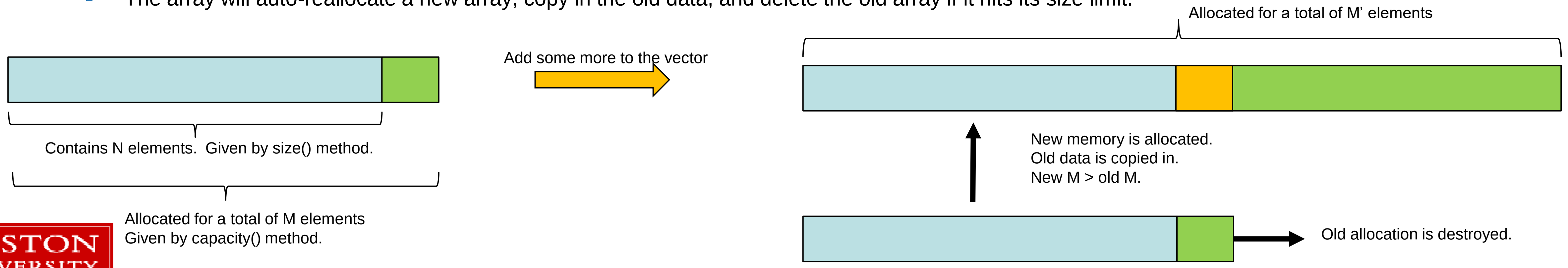
- The implementation is hidden and the necessary code for reverse() is generated from templates at compile time.

vector<T>

- A very common and useful class in C++ is the vector class. Access it with:

```
#include <vector>
```

- Vector has many methods:
  - Various constructors
  - Ways to iterate or loop through its contents
  - Copy or assign to another vector
  - Query vector for the number of elements it contains or its backing storage size.
- Example usage: `vector<float> my_vec ;`
- Or, create my_vec with storage pre-allocated: `vector<float> my_vec(50) ;`
- Hidden from the programmer is the backing store:
  - An array allocated in memory that is at least the size of the number of elements you have added or requested.
  - The array will auto-reallocate a new array, copy in the old data, and delete the old array if it hits its size limit.



BOSTON UNIVERSITY

Destructors

- vector<t> can hold objects of any type:
  - Primitive (aka basic) types: int, float, char, etc.
  - Objects: string, your own classes, file stream objects (ex. ostream), etc.
  - Pointers: int*, string*, etc.
- When a vector is destroyed:
  - If it holds primitive types or pointers it just deallocates its backing store.
  - If it holds objects it will call each object’s destructor before freeing its backing store.

BOSTON UNIVERSITY

vector<t> with objects

- If a vector<MyClass> has had some elements added to it the objects can be accessed via the vector using index notation, iterators, via the at() method, etc.
  - *vec.at(2)* is equivalent to *vec[2]* except that *at(2)* double checks the size of the vector before returning the value.

```
// a vector with memory preallocated to
// hold 1000 objects.
vector<MyClass> my_vec(1000);

// Now make a vector with 1000 MyClass objects
// that are initialized using the MyClass constructor
vector<MyClass> my_vec2(1000,MyClass(arg1,arg2));

// Access an object's method.
my_vec2[100].some_method() ;
// Or a member
my_vec2[10].member_integer = 100 ;


// Or in a loop. const prevents the elem
// reference variable from editing the object
// it refers to.
for (const auto &elem : my_vec2) {
        cout << elem.some_method() << endl ;
}

// Or...without the reference elem is now a
// COPY of the vector element!!
for (auto elem : my_vec2) {
        cout << elem.some_method() << endl ;
}
```

BOSTON UNIVERSITY

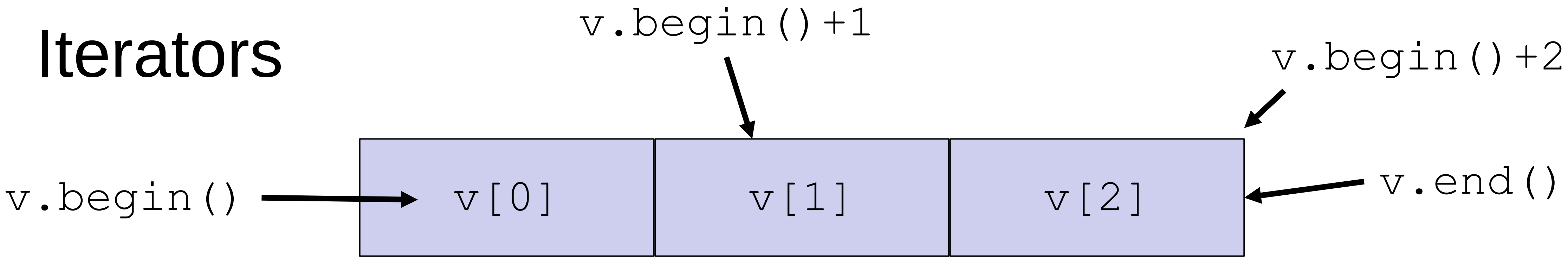
Looping

```
for (int index = 0 ; index < vec.size() ; ++index)
{
        // ++index means "add 1 to the value of index"
        cout << vec[index] << " " ;
}
```

- Loop with a “for” loop, referencing the value of vec using brackets.
- 1st time through:
  - index = 0
  - Print value at vec[0]
  - index gets incremented by 1
- 2nd time through:
  - Index = 1
  - Etc
- After last time through
  - Index now equal to vec.size()
  - Loop exits
- Careful! Using an out of range index will likely cause a memory error that crashes your program.
- Note we call the size() method on every iteration.

BOSTON UNIVERSITY

Iterators



- Iterators are generalized ways of keeping track of positions in a container.
- 3 types: forward iterators, bidirectional, random access
- Forward iterators can only be incremented (as seen here)
- Bidirectional can be added or subtracted to move both directions
- Random access can be used to access the container at any location

Looping

```
for (vector<int>::iterator itr = vec.begin(); itr != vec.end() ; ++itr)
{
      cout << *itr << " " ;
      // iterators are pointers!
}
```

- Loop with a “for” loop, referencing the value of vec using an **iterator** type.
- `vector<int>::iterator` is a type that iterates through a vector of int’s.
- 1st time through:
  - itr points at 1st element in vec
  - Print value pointed at by itr: *itr
  - itr is incremented to the next element in the vector
- Iterators are very useful C++ concepts. They work on any STL container!
  - **No need to worry about the # of elements!**
  - Exact iterator behavior depends on the type of container but they are guaranteed to always reach every value.
- Note we are now retrieving the end iterator at every loop to see if we’ve reached it: vec.end()

Looping

```
for (auto itr = vec.begin(), auto vec_end = vec.end() ; itr != vec_end ; ++itr)
{
        cout << *itr << " " ;
}
```

- Let the *auto* type asks the C++ compiler to figure out the iterator type automatically.
  - This is MUCH easier code to read.
- An extra modification: Assigning the vec_end variable avoids calling vec.end() on every loop.
  - This is faster, for when it matters.

Looping

```
for( auto &element : vec)
{
        cout << element << " " ;
}
```

- Another iterator-based loop: iterator behavior is handled automatically by the compiler and the element type is *also* handled by the compiler.
- Uses a reference so there is no annoying pointer syntax.
- Option: use a ***const auto*** type so the reference to the element can't alter the vector.
- Less typing == less chance for program bugs!
  - The compiler doesn't mind doing extra work.

- Performance considerations:
  - The iterators are general purpose and safer than bracket notation but a wee bit slower. Usually safety and less debugging effort wins over micro-optimizing your code.
  - Using them allow you to substitute different container types if you need to as not every container supports the bracket notation.
  - …meaning you can write a function that takes in begin and end iterators and works on any STL container type!

BOSTON UNIVERSITY

Using vector<> with our Shape classes

- Open the C::B project Part 4/STL Containers
- This has some worked examples using the vector<> class.
- It includes the Shape class hierarchy worked out in Part 3 of the tutorial.
- Let's debug our way through the code to see how a vector<> handles objects that are members of a class hierarchy.
- Introduced in the code: some memory allocations and management.

Some OOP Guidelines

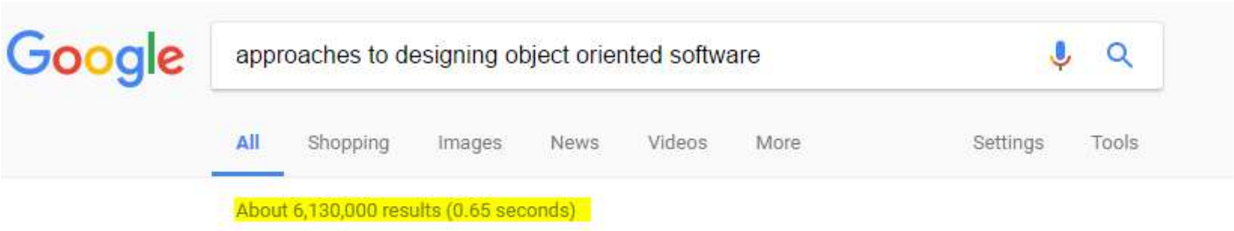
- Here are some guidelines for putting together a program using OOP to keep in mind while getting up and running with C++.

- Keep your classes simple and single purpose.
- Logically organize your classes to re-use code via inheritance.
- Use interfaces in place of multiple inheritance
  - Don’t make your life harder while trying to learn the language.
- Keep your methods short
  - It’s better to have many descriptive methods that do little things than giant methods that do lots of things.
  - This also makes for easier debugging.

- Follow the KISS principle:
  - “Keep it simple stupid”
  - “Keep it simple, silly”
  - “Keep it short and sweet”
  - “Make Simple Tasks Simple!” – Bjarne Stroustroup
  - “Make everything as simple as possible, but not simpler” – Albert Einstein

Putting your classes together

- Effective use of OOP demands that the programmer think/plan/design first and code second.
- There is a large body of information on this topic:



- As this is an academic institution your code may:
  - Live on in your lab long after you have graduated
  - Be worked on by multiple researchers
  - Adapted to new problems you haven't considered
  - Be shared with collaborators
- For more structured environments (ex. a team of professional programmers) there exist concepts like SOLID that seek to create OOP code that is maintainable and easily debuggable over time:
  - https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
  - …and there are many others.

Keep your classes simple

- Avoid “monster” classes that implement everything including the kitchen sink.
- Our Rectangle class just holds dimensions and calculates its area.
  - It cannot print out its area, send email, draw to the screen, etc.
- Two standard approaches to help with this are below.

- Single responsibility principle:
  - Every class has responsibility for one piece of functionality in the program.
  - https://en.wikipedia.org/wiki/Single_responsibility_principle
  - Example:
    - An Image class holds image data and can read and write it from disk.
    - A second class, ImageFilter, has methods that manipulate Image objects and return new ones.

- Resource Allocation Is Initialization (RAII):
  - A late 80’s concept, widely used in OOP.
  - https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization
  - Resources in a class are created in the constructor and released in the destructor.
    - Example: opening files, allocating memory, etc.
  - Therefore resources are guaranteed to be available before the object is used and will be properly handled during program errors.

BOSTON UNIVERSITY

C++ Libraries

- There are a ***LOT*** of libraries available for C++ code.
  - Sourceforge alone has >7300
  - https://sourceforge.net/directory/language:cpp/os:windows/?q=library
- Before jumping into writing your code, consider what you need and see if there are libraries available.

- Many libraries contain code developed by professionals or experts in a particular field.
- Consider what you are trying to accomplish in your research:
  - A) accomplishments in your field or
  - B) C++ programming?
- Probably (A) but there's nothing wrong with (B), especially if C++ skills will be important to you in the future!

BOSTON UNIVERSITY

Multithreading

- OpenMP
  - Open MP is a standard approach to writing multithreaded code to exploit multiple CPU cores with your program.
  - Fully supported in C++
  - See http://www.openmp.org/ for details, or take an RCS tutorial on using it.
- Intel Thread Building Blocks
  - C++ specific library
  - Available on the SCC from Intel and is also open source.
  - Much more flexible and much more C++-ish than OpenMP
  - Offers high performance memory allocators for multithreaded code
  - Includes concurrent data types (vectors, etc.) that can automatically be shared amongst threads with no added effort for the programmer to control access to them.
  - If you want to use this and need help email help@scc.bu.edu

Math and Linear Algebra

- Eigen
  - http://eigen.tuxfamily.org/index.php?title=Main_Page
  - Available on the SCC.
  - “Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.”
- Armadillo
  - http://arma.sourceforge.net/
  - Available on the SCC.
  - “Armadillo is a high quality linear algebra library (matrix maths) for the C++ language, aiming towards a good balance between speed and ease of use. Provides high-level syntax (API) deliberately similar to Matlab.”
  - Also see *matlab2cpp* (https://github.com/jonathf/matlab2cpp), a semi-automatic tool for converting Matlab code to C++ with Armadillo.
  - And also see *PyJet* (https://github.com/wolfv/pyjet), which converts Python and Numpy code to Armadillo/C++ code.
- As nice as the vector<> class is, it’s not going to come close to competing with optimized libraries for handling linear algebra.

Example: Speeding up Python

- Let's take a terrible Python function: a naïve matrix-matrix multiplication implemented with lists.
- Implemented as a method in a barebones Python Matrix class.

- A C++ version should be faster (even keeping the naïve multiplication algorithm).
- Ideally we'd like to drop the C++ code into our Python script.

- For fun, compare against the same routine run with numpy (uses optimized C).

- Python code:
- Please do not ever use this terrible in your own code, this is for demonstration purposes only!

```
class Matrix:
    ''' A barebones implementation of a square matrix.  Note how
        slow the matrix-matrix multiplication is!  '''

    def __init__(self, size=100):
        self.size=size
        self.matrix=[[0.0 for x in range(self.size)] for y in range(self.size)]


    def multiply(self,mat):
        ''' Multiply this matrix with another. Return a new matrix.
            Assume incoming matrix is the same size.'''
        # Allocate an output matrix.
        val = Matrix(self.size)
        for i in range(self.size):
            for j in range(self.size):
                for k in range(self.size):
                    # No such thing as private data in Python classes!
                    val.matrix[i][j] += self.matrix[i][k] * mat.matrix[k][j]
        return val

    def __str__(self):
        ''' Make this printable '''
        strval=''
        for i in range(self.size):
            strval += str(self.matrix[i]) + '\n'
        return strval
```

C++ Class

- matrix_plugin.h

- An equivalent C++ class.
- Stores its matrix as a vector of vectors.
- Uses the same algorithm.
- Will it be faster?

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

class Matrix_Plugin {
    public:
        Matrix_Plugin(const int) ;
        // Don't allow for an empty constructor,
        // only create if a matrix size is given.
        //Matrix_Plugin()=delete ;

        // Duplicate the Python methods for code compatibility

        Matrix_Plugin multiply(const Matrix_Plugin &mat) ;

        // Add an extra "get" method for the size. This lets
        // outside code read the size from the private member.
        int get_matrix_size() ;

        // In order to make this work like the Python class it
        // would be nice to use mymatrix[][] but there is no [][] operator
        // in C++. The () operator can be used instead but that gets a little
        // complicated for this example.  So use a get/set combo
        double get_val(const int row, const int col) ;
        void set_val(const int row, const int col, double val) ;

    private:
        // Need to store the matrix...let's use an STL vector
        // A vector of vectors of doubles will do the trick.
        vector< vector<double> > m_matrix ;

        // And store the matrix size
        int m_matrix_size ;
} ;
```

BOSTON UNIVERSITY

- matrix_plugin.cpp

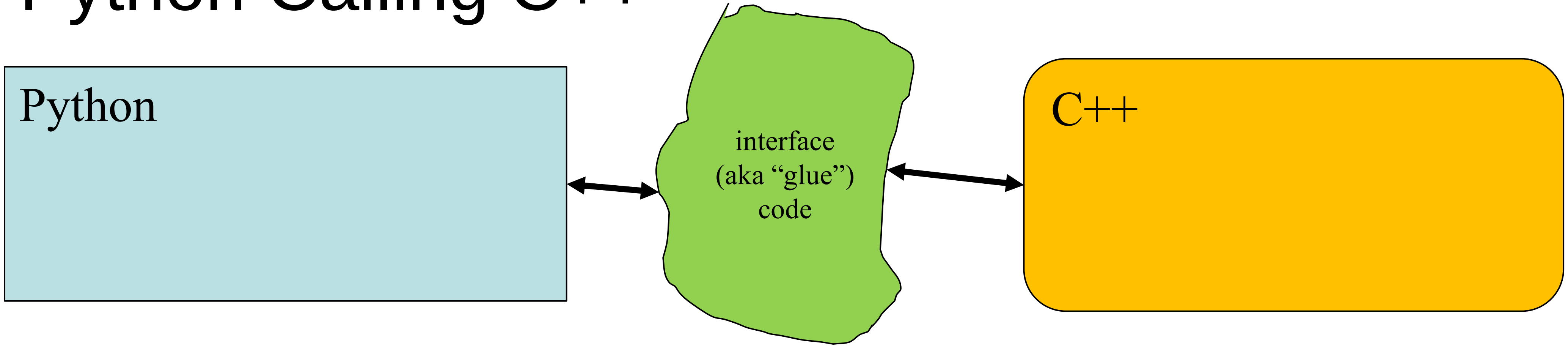
- This is a snippet, showing just the matrix multiplication method. Exact same implementation as in Python.

```
#include "matrix_plugin.h"

Matrix_Plugin Matrix_Plugin::multiply(const Matrix_Plugin &mat) {
    // Implement the exact same algorithm as in Python
    Matrix_Plugin C(m_matrix_size) ;
    for (int i = 0 ; i < m_matrix_size ; ++i) {
        for (int j = 0 ; j < m_matrix_size ; ++j) {
            for (int k = 0 ; k < m_matrix_size ; ++k) {
                C.m_matrix[i][j] += m_matrix[i][k] * mat.m_matrix[k][j] ;
            }
        }
    }
    return C ;
}
```

BOSTON UNIVERSITY

Python Calling C++



- Python requires interface or glue code to be written to translate back and forth between the Python interpreter and the C++ code.
- This interface code is written in C.
- It is a fair amount of labor to write this code and it has to be modified any time the C++ code is changed.
- If only it could be auto-generated…

SWIG – Software Wrapper Interface Generator

- Fortunately there is more than one way to generate the interface code automatically!
- SWIG is a popular and well-tested tool.
  - http://www.swig.org/
  - Can generate wrappers for C and C++ code so it can be called in Python, Perl , Java, R, and ~20 other scripting languages
- An alternative is the Boost.Python library
  - http://www.boost.org/doc/libs/1_63_0/libs/python/doc/html/index.html
  - More powerful and more flexible than SWIG.
  - **Much** steeper learning curve.
- To finish the tutorial let's walk through wrapping faster C++ code with SWIG and plugging it into Python.

SWIG interface file

- File: matrix_plugin.i

```
%module matrix_plugin

%{
    #include "matrix_plugin.h"
%}
%include "matrix_plugin.h"
```

- SWIG needs an interface file, which tells it what parts of the C++ code it should create wrappers around for Python.
- In this case it's pretty minimal. The requirements for this file are in the SWIG documentation but for more straightforward C++ code you just need the header files.

- The C++ class will appear as a Python class.
- The SCC build script is to the right.
- The generated Python interface code is much longer than a human would write at 3661 lines (but it works)!

```
module load swig
module load python/2.7.11
module load gcc/6.2.0
# The matrix_plugin.i is the SWIG interface file.
swig -c++  -python matrix_plugin.i

# SWIG has produced a Python interface file:
#  matrix_plugin_wrap.c

# Compile the C++ plugin and the SWIG interface files
g++ -O3 -fPIC -std=c++11 -c matrix_plugin.cpp
g++ -O3 -fPIC -std=c++11 -c -I$SCC_PYTHON_INCLUDE/python2.7 \
       matrix_plugin_wrap.cxx
# Link into a shared library
g++ -shared matrix_plugin.o matrix_plugin_wrap.o -o _matrix_plugin.so


# In Python now do:
#  import matrix_plugin
#  matrix_plugin.Matrix_Plugin(mat_sz)
# to use the new C++ class!
```

BOSTON UNIVERSITY

How does it compare in speed?

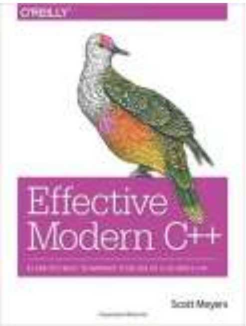
- Compare: original terrible Python code, SWIG'd C++ duplicate code, and numpy.
- Run on scc2.bu.edu
- Time is in seconds.
- Lesson 1: SWIG and C++ are faster than Python!

Matrix Size	Bad Python	C++ Conversion	Numpy
100x100	0.387	0.0015	6.89e-05
200x200	3.35	0.014	0.0002
300x300	11.01	0.049	0.0004
500x500	58.6	0.29	0.0012

- Lesson 2: a crummy algorithm always loses to an optimized algorithm.

BOSTON UNIVERSITY

Some Web and Print Resources



- **C++ Primer (5th Edition)** by Stanley B. Lippmanm,Josée Lajoie, Barbara E. Moo
  - A well-regarded book for anyone learning C++
- **Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14 (1st Edition)** by Scott Meyers
  - Not a beginner's book, but excellent once you feel confident in C++
- **The C++ Standard Library: A Tutorial and Reference (2nd Edition)** by Nicolai M. Josuttis
  - An excellent reference on the STL.

- **http://www.cplusplus.com/**
  - Has tutorials, articles, C++ information, reference materials, and a forum.
  - The reference is **excellent** with clear explanations and good example code.
- http://en.cppreference.com/w/cpp
  - A highly detailed and technical reference. Useful when cplusplus.com doesn't describe enough of the underlying behavior of things in the C++ language and STL for you.
- https://isocpp.org/faq
  - The C++ Super FAQ
  - A great resource!
- https://www.tutorialspoint.com/cplusplus/index.htm
  - Tutorialspoint has tutorials for multiple languages.

BOSTON UNIVERSITY

C++ on the SCC

- Gnu **g++** compiler. For C++11 use at least version 4.9.2
  - Ex.:

```
module load gcc/6.2.0
g++ -o myprog -std=c++11 myfile.cpp
```

- Intel **icc** compiler. Both available versions (2015 and 2016) support C++11.
  - Ex.:

```
module load intel/2016
icc -o myprog -std=c++11 myfile.cpp
```

- LLVM **clang++** compiler. Use at least version 3.9.0
  - Ex.:

```
module load llvm/4.0.0
clang++ -o myprog -std=c++11 myfile.cpp
```

Tutorial Conclusion

- Topics covered:
  - Some basic C++ syntax
  - OOP concepts: encapsulation, abstraction, inheritance, polymorphism
  - Classes:
    - Syntax
    - Private / protected / public access
    - Methods and members
    - Inheritance
    - Use of the *virtual* keyword
  - Basics of templates and the STL
  - Use of an IDE (Code::Blocks) to assist with development and debugging
  - Example of accelerating Python with C++

Topics Not Covered

- Where to start?!
  - More C++ syntax
    - If/else, other types of loops
  - Manual memory management
  - File I/O
  - Pointer intricacies
  - Design patterns
    - More ways to organize OOP code
  - Multithreading
  - Template metaprogramming
    - Wherein templates are used not just to adapt to types but to generate code for you.
    - This is the basis of libraries like Armadillo.
    - Really advanced C++.
    - In C++ the template system was accidentally discovered to be ***Turing complete***, meaning it can compute anything computable (like C++ itself)…in other words it's like having a separate programming language embedded in C++!

Your thoughts please!

- The main tutorial goal was to introduce C++ and to cover the main reason for choosing it over competitors like FORTRAN or C: object-oriented programming.

- The focus has been on developing classes and understanding the OOP concepts underlying the approach.

- This is the first time that a C++ tutorial has been offered by RCS. What would you change? Add? Remove?