

C++ Tour

For : COP 3330.
Object oriented Programming (Using C++)
<http://www.cs.fsu.edu/~piyush/teach/3330>

Piyush Kumar



Administrative Trivia

About me:

Piyush Kumar

Phone: 645-2355

Email: piyush@cs.fsu.edu

Office hours: Tuesday 4:30 to 5:30.

TAs: Biswas Parajuli,

Daniel Mock,

James Parsons

More details on the course information sheet.



Administrative Trivia

TAs:

Biswas Parajuli, (Sec: 15, 17)

Daniel Mock, (Sec: 10, 16)

James Parsons. (Sec: 2)

Office hours will be held in MCH 315A. Subject to change.
More details on the course information sheet.



Your ID

You have been assigned an ID on the Blackboard. You should know your ID.

You will use your FSU ID to setup your bitbucket account.

<https://bitbucket.org/>




Announcement

- First Quiz : Now
- Revise your previous C/C++ material.
 - Your C++ background will be tested soon.
- Your first assignment is online.
 - Will be due on Tuesday, 1/17/17.
 - You are required to setup your bitbucket repository by this Thursday. The submission will use the blackboard account.




Announcement

- Blackboard Discussions already setup.
- Make sure you submit the pre-req form today before you leave.
- Bitbucket setup of ssh keys.
 - `ssh-keygen -t rsa -C "fsu email"`
 - Add it to bitbucket ssh-keys (without adding \n)
 - `hg clone project1`




C++ Tour




Why learn C++?

- Ubiquitous
- Object Oriented
 - Easier large scale projects
 - Resuability
- High Performance




C++ Features

- Supports data security
- Prevents accidents with data
- Helps code reuse.
- Lets you use operators the way you like.
- Allows multiple functions/operators with the same name.



When to use C++?

- Large projects
- System applications
- Graphics
- Data Structures
- Speed is an issue?
- Changes in functionality required.
- Need exceptions and error handling?
- Want to speed up your scripts?



When not to use C++?

- Small system programs.
- Fast prototyping.
- Web-based applications (Perl/Python)



Important definitions

- Algorithm: Ordered set of actions to accomplish a certain task.
- Program: Implementation of algorithms.
- Compiler, function, library, bug.
- Variables, constants.
- Keywords (if, while, for,...)
- Data Types. (long, int, ...)

Compiling/Running programs

- Single source file code:
 - `g++ -g -Wall -std=c++1y simple.cpp -o simple`
- Compilation / Editing Demo
 - `int main() {`
 - `return 0;`
 - `}`

Lynda.com

- Up and running with vi
 - By David D. Levine
- 1.5 hour video course in vim
- Your homework for this weekend is to watch and practice vim

simple.cpp

```
int main() // Function Declaration
// Function body follows

{ // Block of statements begins.

    return 0;

} // Block ends.
```

simple.cpp

- On Unix: `echo $?`
- On Windows: `echo %ERRORLEVEL%`

Simple.s

System dependent
(using `g++ -O -S simple.cpp`)

```
.file "simple.cpp"
.def __main; .scl 2; .type 32; .endef
.text
.align 2
.p2align 4,,15
.globl __main
.def __main; .scl 2; .type 32; .endef
__main:
    pushl %ebp ; Save base pointer
    movl $16, %eax
    movl %esp, %ebp ; Set up stack frame for debugger
    subl $8, %esp ; Save space on the stack
    andl $-16, %esp ; Align stack pointer
    call __alloca ; Platform dependent call
    call __main ; Platform dependent call
    leave ; free space, pop ebp, esp...
    xorl %eax, %eax ; zero eax
    ret ; return control to calling procedure.
```

Simple.s (using VC++ 05 compiler)

```
/(
00411360 push ebp
00411361 mov ebp,esp
00411363 sub esp,0C0h
00411369 push ebx
0041136A push esi
0041136B push edi
0041136C lea edi,[ebp-0C0h]
00411372 mov ecx,30h
00411377 mov eax,0CCCCCCCCh
0041137C rep stos dword ptr es:[edi]
// return 0;
0041137E xor eax,eax
//)
00411380 pop edi
00411381 pop esi
00411382 pop ebx
00411383 mov esp,ebp
00411385 pop ebp
00411386 ret
```

G++ Compilation.

- preprocessing (to expand macros)
 - Try "cpp simple.cpp > simple.i"
- compilation (from source code to assembly language)
 - Try "g++ -Wall -S simple.i"
- assembly (from assembly language to machine code)
 - Try "as simple.s -o simple.o"
- linking (to create the final executable)
 - Try "gcc simple.o"
 - Equivalent to "ld -dynamic-linker /lib/ld-linux.so.2 /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/gcc-lib/i686/3.3.1/crtbegin.o -L/usr/lib/gcc-lib/i686/3.3.1 hello.o -lgcc -lgcc_eh -lc -lgcc -lgcc_eh /usr/lib/gcc-lib/i686/3.3.1/crtend.o /usr/lib/crtn.o"

\$ file ./a.exe → Identify type of file generated (gcc on windows).
./a.exe: PE executable for MS Windows (console) Intel 80386 32-bit

Introduction to C++

Organization

- C++ = C + objects + ...
- Main Theme : Objects
- Class Vs Object
- Data Hiding and Abstraction
- Encapsulation
- Inheritance
- Operator and Function Overloading
- Polymorphism.
- Generic Programming

C++ is a superset of C.

- Features present in C are also present in C++
- C++ is a object oriented programming language (C++ = c + objects + ...)
- OOP is about **objects** which contain data and functions to manipulate that data.
- A single unit that contains data and functions that operate on the data is called an **object**.

Main Theme: objects

- C++ enables you to focus on discrete objects that contain both data and functions to manipulate that data.
- Instead of data and functions as separate entities as was the case in C.
- Application = Collection of Objects.
 - Makes complex programs easy to code.

OOP

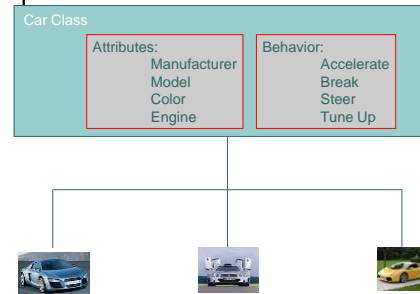
- Based on concept of objects and classes.
- Objects: Represent entities with related state and behaviour
 - Instances of a class
- Classes: Define common characteristics of similar objects.

Objects/Classes

- Objects are reusable self-contained programming modules with data and functions.
- Classes are blue-print for objects with common properties, attributes, operations and behaviors.

The principal building blocks of OO programs are classes and objects.

Object / Class Example



Objects: Instantiations of classes.

Another Object / Class example

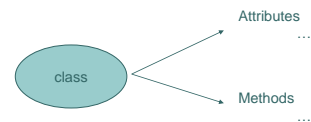
```
class Point {
    int _x, _y; // point coordinates

    public: // begin interface section
        void setX(const int val);
        void setY(const int val);
        int getX() { return _x; }
        int getY() { return _y; }
};

Point tpoint; // defines an object tpoint
```

Fundamental building block of OOP: A **Class**

- A **class** defines a data type, much like a **struct** would be in C.
- A class is a source code for an object (hence it has both data+**member** functions)

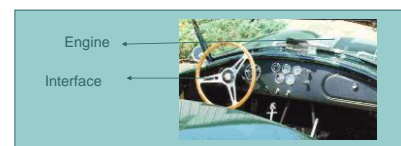


Fundamental building block of OOP: A **Class**

- You can imagine that **int** is a class that has member functions called **operator++**, etc.
- An object usually means an instance of a class
 - After the declaration **int i**; we say that "**i** is an object of type **int**."

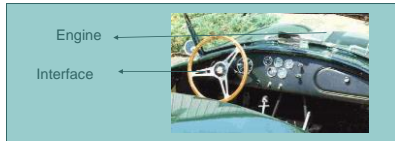
Data Hiding

- Objects contain information
- Only part of the information contained in the object might be presented to the user.
- Rest is concealed.



Data Hiding

- Internal dynamics are not visible to the user.
- Data hiding is done by using the “private” keyword in the class definition.



Data Hiding

- Allows data to be accessed by certain functions

```
class Point { // private
    // concealed info
    int _x, _y; // point coordinates
public: // begin interface section

    void setX(const int val);
    void setY(const int val);
    int getX() { return _x; }
    int getY() { return _y; }

};

Point tpoint; // defines an object tpoint
tpoint._x = 5; // ILLEGAL
```

Data Abstraction

- DA is a programming technique where one separates the interface from the implementation
- Class designer worries about the interface
- Programmers implement

```
String s; s.rfind("\");
```

Data Encapsulation

- Combine lower level elements to form a new higher level entity.
- Grouping of attributes and behaviors to form an object.

C++ **vector** type is an example of both data abstraction and Data encapsulation

OOP : Inheritance



- Many classes have common attributes.
- These classes can be arranged in a hierarchy.
- Inheritance** enables you to reuse code and add data and functionality without making any changes to the existing code.
 - For example, objects can *inherit* characteristics from other objects.

Inheritance

- Children inherit traits from their parents.
- When a class is inherited all the functions and data member are inherited, although not all of them will be accessible by the member functions of the derived class.

Inheritance

- protected/public members of the base class are accessible to the derived class
- private members are not.

```
class vehicle {
protected:
    char colname[20];
    int number_of_wheels;
public:
    vehicle();
    ~vehicle();
    void start();
    void stop();
    void run();
};
```

Super class or Base class

```
class Car: public vehicle {
protected:
    char type_of_fuel;
public:
    Car();
};
```

Subclass or derived class

Inheritance

- Derived classes are specialized form of their base class.
- Abstract class** : A base class that is undefined and unimplemented.

Multiple definitions...?

- C++ allows for the same function (or overloaded operator) to have multiple definitions.

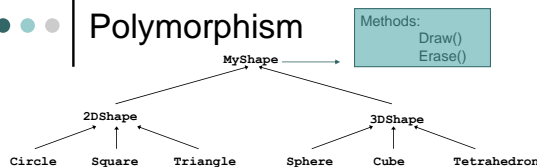
```
Swap(int& l, int& j);
Swap(string& l, string& j);
Swap(double& l, double& j);
```

a.k.a. **Function overloading**.

Operator Overloading

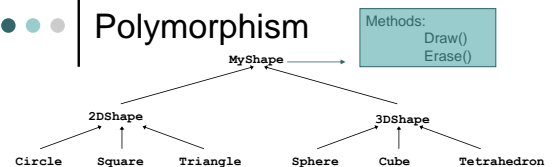
- C++ gives you the power to define operators on user defined types.
- Example:
 - MyMatrix m = m1 + m2; // cant do this in C.
- Overloading is a type of polymorphism.
- Not all operators can be overloaded in C++.

Polymorphism



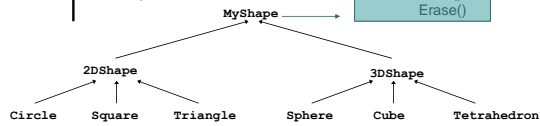
- Base Class MyShape establishes the common interface to anything inherited from MyShape.
- All shapes can be drawn and erased.

Polymorphism



- Overriding: When a child class extends the functionality of its parent class.
- Polymorphism allows the programmer to implement different draw/erase methods for derived classes.
- Sphere and Cube have the same function draw() with different functionality.

Polymorphism



- No matter what shape the object is, one can use the “draw” method to draw it correctly.

Polymorphism

- Using operators and functions in different ways depending on what they are operating on is called polymorphism.
 - Static
 - Dynamic
- These concepts builds upon encapsulation and inheritance.

Generic Programming

- Type independent code
- Write code for one generic type and use it for multiple types.

```
Without Generic Programming:  
void swap(int &x, int &y) { int tmp = x; x = y; y = tmp; }  
void swap(long &x, long &y){ long tmp = x; x = y; y = tmp; }  
...  
  
With Generic Programming  
template <typename T>  
void swap(T &x, T &y) { T tmp = x; x = y; y = tmp; }
```