# Iteration 2 Submission : Due THURSDAY at 11:55pm. PULLING at midnight!

ITEM	SUBMISSION
Priority 1, 2, 3 Implemented Iteration 2 folder!!!	Merge into master branch Tag with v.2.0
Bug Report	<i>"BugReport.</i> *" in /docs OR as part of Doxygen along with todo list
Design Document Major design decisions with justification including other options that you considered.	"DesignDocument.*" in /docs
UML to match your code	UML.pdf in /docs
Mainpage.h (do you need to change this?) COMPILE YOUR DOXYGEN. Don't leave stuff around.	Do not submit the html folder. We will compile.
Google Style Compliant : Check Naming Conventions!	Run cpplint – no errors!
Github Usage : branches, issues, regular commits, good messages!!	We look at history and a usage report
Google Tests : tests for sensors.	Visual inspection – not passing.

### **Bug Report**

Identify all known bugs. Describe the bug. If you don't have any known bugs, report that.

Bugs are not todo lists – they are for implementation that is not working as required or desired!

ALSO,

if you do have incomplete functionality include it here making it clear what is not implemented and what is buggy.

OR,

If you are doing this in doxygen, use @bug and @todo.

You will be docked points if we find a bug that you did not report.

## **Templates and Containers**

### CS3081 Program Design and Development

## Polymorphism

**Polymorphism**: generally defined as "the ability to create a variable, a function, or an object that has more than one form." The result is that you get different behavior (i.e. different pieces of code are executed) depending on the type of object or objects that are being acted upon.

- **Operator Overloading**: One operator can be applied to different types.
- Method Overriding (Ad-hoc polymorphism): Derived class redefining base class method.
- **Method Overloading (Ad-hoc polymorphism)**: Multiple function definitions with different parameter lists.
- **Subtype Polymorphism**: Upcasting derived class object can be used in place of base class object.
- **Parametric Polymorphism**: Templates one function with same behavior across multiple types. (Stack of ints, strings, ClassA, ... )

cite: Wikipedia and http://www.catonmat.net/blog/cpp-polymorphism/

### Templates

"Inheritance and composition provide a way to reuse object code. The *template* feature in C++ provides a way to reuse *source* code."

A way for you to write generic, type-less code.

Example:

class IntegerArray and class FloatArray and class CharArray

instead template<class Type> class Array

# Strong Type Checking\*

- A type can be defined as
  - a set of permitted values and
  - a set of operations permitted on these values
- Important operations in programming languages related to types:
  - defining a new type
  - declaring variables to be of a certain type
  - checking that no type errors can occur type checking

\* from Gopalan Nadathur via Eric VanWyk

# Type Checking

- Types and type checking help us avoid type errors.
- Type error occurs when sequence of bits that represent one kind of data is interpreted as another kind of data. (e.g. ...
  - reading sequence of bits that store an int value as a char \*
  - reading the representation of an object as a string
- The nature of research in type systems:
  - preserve strong typing while
  - making the type system more liberal.
  - That is, continue to disallow any bad programs but allow more good programs.

A major advance: parametric polymorphism.

\* from Gopalan Nadathur via Eric VanWyk

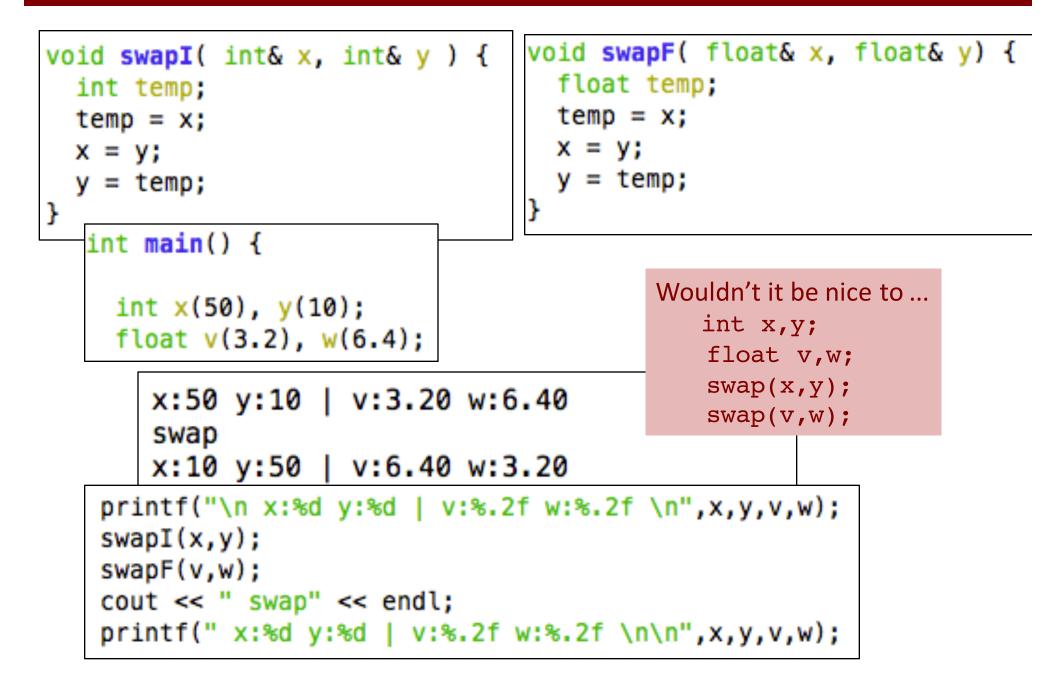
To implement a function that has parameterized types a compiler can:

- 1. Create a single function that works on all types (in machine language)
- 2. Determine what types are passed as parameters to the functions and generate a special case function for each of these types. This is what C++ does for template types.\*

### Vector Template

```
std::vector<class ArenaEntity*> entities_;
entities_.push_back(new ArenaEntity(20));
entities_.push_back(new ArenaEntity(25));
for (std::vector<class ArenaEntity*>::iterator ent = entities_.begin();
ent != entities_.end();
++ent ) {-
(*ent)->Print();-
}-
```

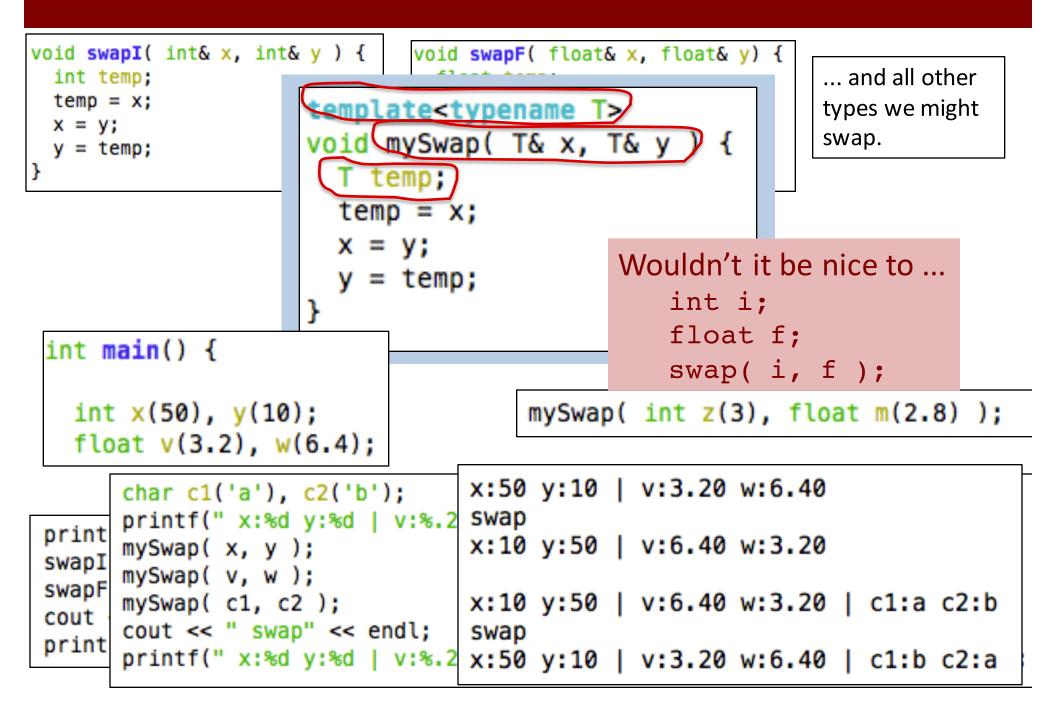
### The Need for Polymorphism



# **Overloaded Functions (Better)**

<pre>void olSwap( int&amp; x, int&amp; y ) {     int temp;     temp = x;     x = y;     y = temp;</pre>	<pre>void olSwap( float&amp; x, float&amp; y ) {   float temp;   temp = x;   x = y;   y = temp; }</pre>
(	Wouldn't it be nice to write 1 piece of code for all types! x:50 y:10   v:3.20 w:6.40 overloaded swap x:10 y:50   v:6.40 w:3.20
<pre>print printf("\n x:%d y:%d   v:%.2f w:%.2f \n",x,y,v,w); swapI olSwap( x, y ); swapF olSwap( v, w ); cout cout &lt;&lt; " overloaded swap" &lt;&lt; endl; print printf(" x:%d y:%d   v:%.2f w:%.2f \n",x,y,v,w);</pre>	

### Generic Swap with Templates

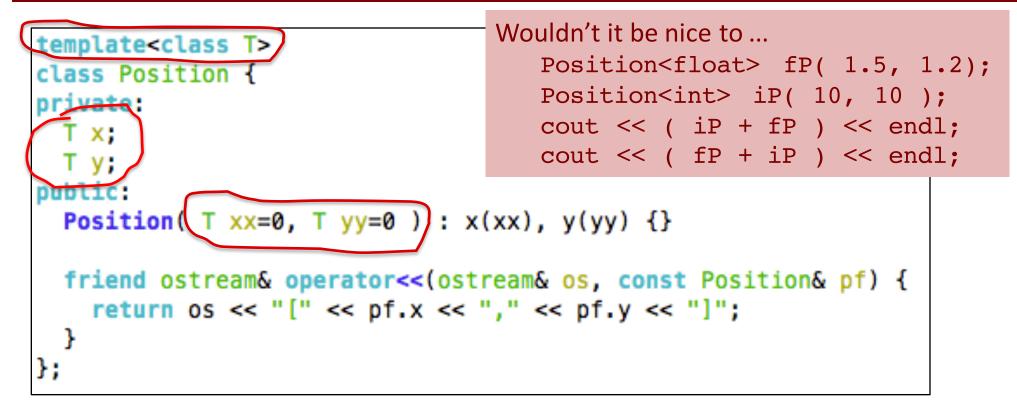


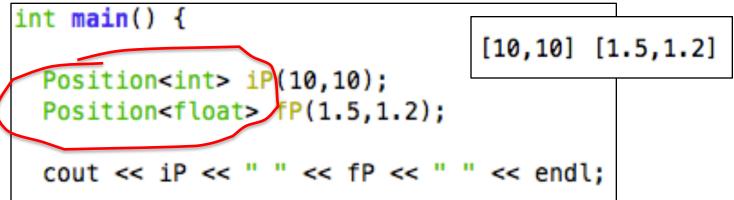
#### **Templates and Classes**

```
class Position {
private:
    int x;
    int y;
public:
    Position( ) : x(0), y(0) {}
    Position( int inX, int inY ) : x(inX), y(inY) {}
```

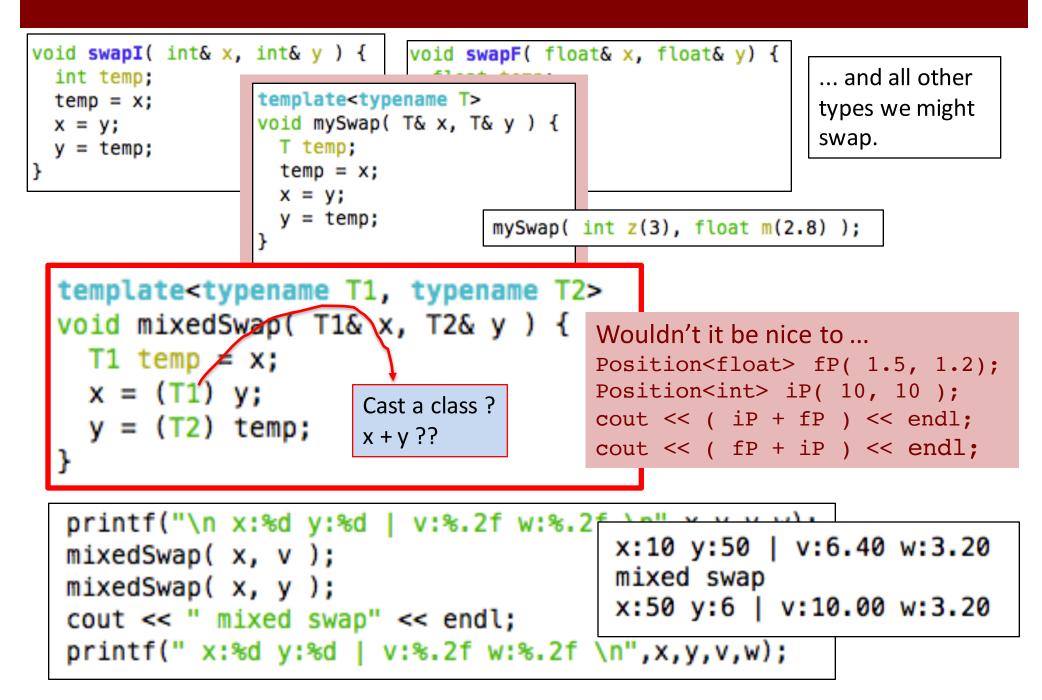
```
class PositionFloat {
private:
   float x;
   float y;
public:
   PositionFloat(float inX=0.0, float inY=0.0) : x(inX), y(inY) {}
```

### **Classes of Different Types**





#### Mixed Types in Swap



### Array

Define a class called Array that holds N objects of type T template<class T, int N>

Overload the operator [] for indexing the array. Be sure to check bounds when indexing

### **C++ Containers**

#### Sequence containers

Sequence containers implement data structures which can be accessed sequentially.

array (since C++11)	static contiguous array (class template)
vector	dynamic contiguous array (class template)
deque	double-ended queue (class template)
<pre>forward_list (since C++11)</pre>	singly-linked list (class template)
list	doubly-linked list (class template)

#### **Container adaptors**

Container adaptors provide a different interface for sequential containers.

stack	adapts a container to provide stack (LIFO data structure) (class template)
queue	adapts a container to provide queue (FIFO data structure) (class template)
priority_queue	adapts a container to provide priority queue (class template)

<u>http://en.cppreference.com/w/cpp/container</u> (there are other containers)

# Built-In Functionality of Containers

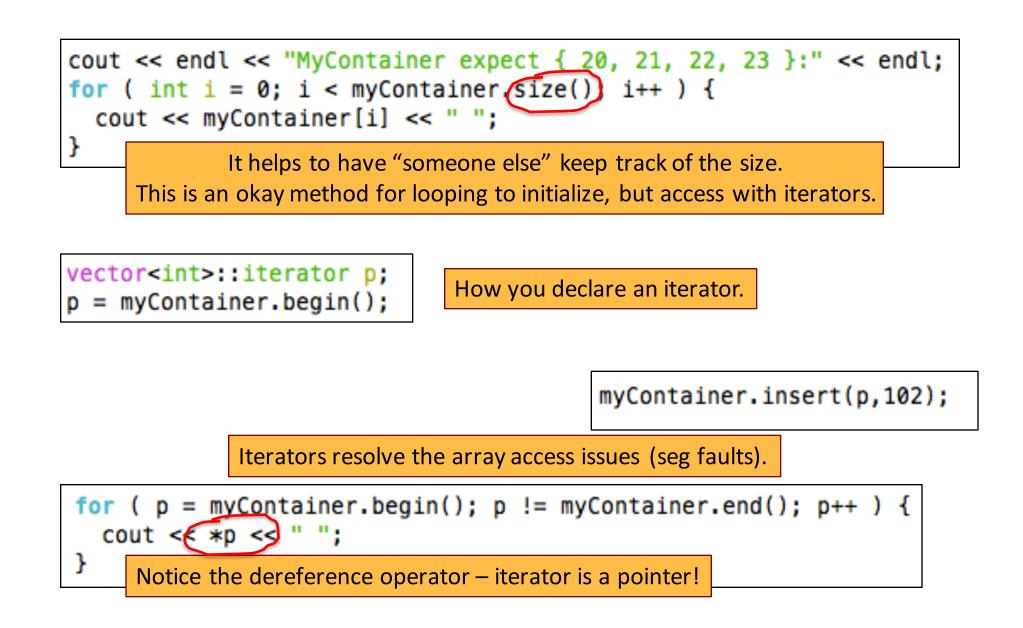
Member func	tions	Modifiers	
(constructor)	constructs the vector (public member function)	clear	clears the contents (public member function)
(destructor)	destructs the vector (public member function)	insert	(public member function)
operator=	assigns values to the c (public member function)	emplace(C++11)	constructs element in-place (public member function)
assign	assigns values to the c (public member function)	erase	erases elements (public member function)
Element access		push_back	adds elements to the end (public member function)
at	access specified elem (public member function)	emplace_back(C++11)	constructs elements in-place at th (public member function)
operator[]	access specified elem (public member function)	pop_back	removes the last element (public member function)
front	access the first eleme (public member function)	resize	changes the number of elements s (public member function)
back	access the last eleme (public member function)	swap	swaps the contents (public member function)

Iterators	
begin	returns an iterator to the beginning
cbegin	(public member function)
end	returns an iterator to the end
cend	(public member function)

## Modifying a Container

- Delete Elements
  - vector.pop\_back(); // remove last element
  - vector.erase( iterator ); // remove element at iterator location
  - vector.clear(); // remove all elements
- Access Elements
  - vector.front(); // first element
  - vector.back(); // last element
  - vector.at(position); // check bounds
  - vector[#];
- Iterator (special "smart" pointers for accessing containers)
  - vector.begin(); // returns iterator to first element
  - vector.end(); // returns iterator to last element

#### Iterators



## **Templates and Containers**

- Templates
  - Compiler creates a class/function for each needed type.
  - Defining (put it all in header)
  - Methods in template must be defined for class.
- Containers
  - For safety and ease.
  - Use iterators (declared as a specific container type iterator).