

# C++ Primer for CS175

Yuanchen Zhu

September 17, 2009

## 1 General C++ Usage

- Headers
  - For headers in the C standard lib: Remove “.h” suffix and add “c” prefix. E.g., `#include <stdio.h>` becomes `#include <cstdio>`
  - C++ standard lib: No suffix/prefix. E.g., `#include <string>`
- C++ namespace: big container of types/functions/variables to avoid name conflict.
  - Everything in the C++ lib is in the `std` namespace. Use `std::` in front of type/function name to reference stuff in the namespace. E.g. to reference a standard C++ string type, use `std::string`;
  - To save typing, can also include entire namespace at once: `using namespace std`; So I won't type `std::` from now on.
- Input/Output: special operators `<<` and `>>`:
  - `printf("Hello World!");` becomes `cout << "Hello World!";`
  - The reason for defining `<<` and `>>` is typesafeness.
- Typecast (again for typesafeness)
  - `static_cast`, `dynamic_cast`, `const_cast`, `reinterpret_cast`

## 2 Class

Think of glorified `struct`.

- Can contain function calls.
- Can contain member variables.
- *Constructor*: functions with the same name as the class. Piece of code called whenever a new instance of the class is instantiated.
  - E.g.: `return Vector3(10.0, 10.0);`
- *Destructor*: piece of code called whenever an instance of the class is deallocated.
- Functions and variables can have different visibility: `public`, `private` (the default), `protected`.
- Inside a member function, `this` variable points to the current object.

### 3 Operator overloading

- Operator is no different from functions, and can be customly defined in C++. This is useful for doing math on customly defined datatypes.
- Later in the class, you have access to `class Vector3`. Given two variables: `Vector3 a, b;`, can write `a += b;` instead of, say, `a.add(b);`
- Operators you're probably going to use:
  - assignment: `a = b;`
  - arithmetics: `a *= b; c = a + b;`
  - indexing: `Vector3 a(10.0, 10.0, 10.0); a[0] += 10.0;`

### 4 Reference

- Behind the scene: like pointer, but does not need `&` and `*`
- Originally, you write:

```
void getX(int *x) { *x = 10; }
getX(&i); // i == 10 now
```
- But now: 

```
void getX(int &x) { x = 10; }
getX(i); // i == 10 now
```
- Useful for passing large object by reference. Often in combination with `const` modifier to make sure object is not modified;
- Can be misused since you no longer know if the variable passed to `getX` will be modified or not. (Unless there's a `const` modifier)

### 5 Templates

- Syntax example: `template <typename T, int n> class Vector {...}`
- Kind of smart macro expansion to avoid code duplication. E.g. `Vector3` is actually an alias for the `Vector` template above with parameter `T = double` and `n = 3`.
- Can be used essentially as a normal class.

### 6 Overloading

- Overloading = same name, multiple definition. But without confusing the compiler.
- Use different parameter list to differentiate.
- Constructors can be overloaded.

### 7 More advanced stuff

- Standard Template Library (STL)
  - Well thought out container library for C++
  - Almost always better than rolling your own.
  - `std::vector<int> a;` instead of `int a[100];`
- Iterators: more on them when we encounter them later in the semester
- OO Stuff: inheritance, polymorphism, etc. Mostly not used in this class.