



## C++ Tutorial

Rob Jagnow

## Overview

- Pointers
- Arrays and strings
- Parameter passing
- Class basics
- Constructors & destructors
- Class Hierarchy
- Virtual Functions
- Coding tips
- Advanced topics

## Pointers

```
int *intPtr;           Create a pointer
intPtr = new int;      Allocate memory
*intPtr = 6837;        Set value at given address

*intPtr → 6837
intPtr → 0x0050

delete intPtr;         Deallocate memory

int otherVal = 5;      Change intPtr to point to
intPtr = &otherVal;    a new location

*intPtr → 5 ← otherVal
intPtr → 0x0054 ← &otherVal
```

## Arrays

### Stack allocation

```
int intArray[10];
intArray[0] = 6837;
```

### Heap allocation

```
int *intArray;
intArray = new int[10];
intArray[0] = 6837;

...

delete[] intArray;
```

## Strings

A string in C++ is an array of characters

```
char myString[20];
strcpy(myString, "Hello World");
```

Strings are terminated with the NULL or '\0' character

```
myString[0] = 'H';
myString[1] = 'i';
myString[2] = '\0';

printf("%s", myString);    output: Hi
```

## Parameter Passing

### pass by value

```
int add(int a, int b) {    Make a local copy of a & b
    return a+b;
}
```

```
int a, b, sum;
sum = add(a, b);
```

### pass by reference

```
int add(int *a, int *b) {    Pass pointers that reference
    return *a + *b;          a & b. Changes made to a
                              or b will be reflected
                              outside the add routine
}
```

```
int a, b, sum;
sum = add(&a, &b);
```

## Parameter Passing

pass by reference – alternate notation

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

## Class Basics

```
#ifndef _IMAGE_H_  
#define _IMAGE_H_
```

Prevents multiple references

```
#include <assert.h>  
#include "vectors.h"
```

Include a library file

Include a local file

```
class Image {
```

```
public:  
    ...
```

Variables and functions  
accessible from anywhere

```
private:  
    ...
```

Variables and functions accessible  
only from within this class

```
};
```

```
#endif
```

## Creating an instance

Stack allocation

```
Image myImage;  
myImage.SetAllPixels(ClearColor);
```

Heap allocation

```
Image *imagePtr;  
imagePtr = new Image();  
imagePtr->SetAllPixels(ClearColor);  
  
...  
  
delete imagePtr;
```

## Organizational Strategy

image.h Header file: Class definition & function prototypes

```
void SetAllPixels(const Vec3f &color);
```

image.C .C file: Full function definitions

```
void Image::SetAllPixels(const Vec3f &color) {  
    for (int i = 0; i < width*height; i++)  
        data[i] = color;  
}
```

main.C Main code: Function references

```
myImage.SetAllPixels(clearColor);
```

## Constructors & Destructors

```
class Image {  
public:  
    Image(void) {  
        width = height = 0;  
        data = NULL;  
    }
```

**Constructor:**  
Called whenever a new  
instance is created

```
    ~Image(void) {  
        if (data != NULL)  
            delete[] data;  
    }
```

**Destructor:**  
Called whenever an  
instance is deleted

```
    int width;  
    int height;  
    Vec3f *data;  
};
```

## Constructors

Constructors can also take parameters

```
Image(int w, int h) {  
    width = w;  
    height = h;  
    data = new Vec3f[w*h];  
}
```

Using this constructor with stack or heap allocation:

```
Image myImage = Image(10, 10);    stack allocation
```

```
Image *imagePtr;  
imagePtr = new Image(10, 10);    heap allocation
```

## The Copy Constructor

```
Image(Image *img) {
    width = img->width;
    height = img->height;
    data = new Vec3f[width*height];
    for (int i=0; i<width*height; i++)
        data[i] = new data[i];
}
```

A default copy constructor is created automatically, but it is usually not what you want:

```
Image(Image *img) {
    width = img->width;
    height = img->height;
    data = img->data;
}
```

## Passing Classes as Parameters

If a class instance is passed by reference, the copy constructor will be used to make a copy.

```
bool IsImageGreen(Image img);
```

Computationally expensive

It's much faster to pass by reference:

```
bool IsImageGreen(Image *img);
```

or

```
bool IsImageGreen(Image &img);
```

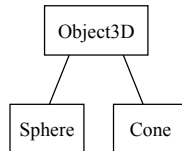
## Class Hierarchy

Child classes inherit parent attributes

```
class Object3D {
    Vec3f color;
};

class Sphere : public Object3D {
    float radius;
};

class Cone : public Object3D {
    float base;
    float height;
};
```



## Class Hierarchy

Child classes can *call* parent functions

```
Sphere::Sphere() : Object3D() {
    radius = 1.0;
}
```

Call the parent constructor

Child classes can *override* parent functions

```
class Object3D {
    virtual void setDefaults(void) {
        color = RED; }
};

class Sphere : public Object3D {
    void setDefaults(void) {
        color = BLUE;
        radius = 1.0 }
};
```

## Virtual Functions

A superclass pointer can reference a subclass object

```
Sphere *mySphere = new Sphere();
Object3D *myObject = mySphere;
```

If a superclass has virtual functions, the correct subclass version will automatically be selected

```
Superclass
class Object3D {
    virtual void intersect(Vec3f *ray, Vec3f *hit);
};

Subclass
class Sphere : public Object3D {
    virtual void intersect(Vec3f *ray, Vec3f *hit);
};

myObject->intersect(ray, hit);
```

Actually calls Sphere::intersect

## The main function

This is where your code begins execution

```
int main(int argc, char** argv);
```

↑  
Number of  
arguments

↑  
Array of  
strings

argv[0] is the program name

argv[1] through argv[argc-1] are command-line input

## Coding tips

Use the `#define` compiler directive for constants

```
#define PI 3.14159265  
#define sinf sin
```

Use the `printf` or `cout` functions for output and debugging

```
printf("value: %d, %f\n", myInt, myFloat);  
cout << "value:" << myInt << ", " << myFloat << endl;
```

Use the `assert` function to test “always true” conditions

```
assert(denominator != 0);  
quotient = numerator/denominator;
```

## “Segmentation fault (core dumped)”

Typical causes:

```
int intArray[10];  
intArray[10] = 6837;
```

Access outside of  
array bounds

```
Image *img;  
img->SetAllPixels(ClearColor);
```

Attempt to access  
a NULL or previously  
deleted pointer

These errors are often very difficult to catch and  
can cause erratic, unpredictable behavior.

## Advanced topics

Lots of advanced topics, but few will be required for this course

- `friend` or `protected` class members
- `inline` functions
- `const` or `static` functions and variables
- *pure* virtual functions  

```
virtual void Intersect(Ray &r, Hit &h) = 0;
```
- compiler directives
- operator overloading  

```
Vec3f& operator+(Vec3f &a, Vec3f &b);
```