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

delete intPtr;
Dealocate memory

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

intPtr

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

pass by reference
int add(int *a, int *b) {
    return *a + *b;
}
int a, b, sum;
sum = add(&a, &b);

pass by reference
Make a local copy of a & b
Pass pointers that reference a & b. Changes made to a or b will be reflected outside the add routine
Parameter Passing

pass by reference – alternate notation

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

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

Class Basics

```c
#include <cassert>
#include "vectors.h"
class Image {
    public:
        ...
    private:
        ...
};
```

Organizational Strategy

```c
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);
```

Creating an instance

Stack allocation

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

Heap allocation

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

Constructors & Destructors

```c
class Image {
    public:
        Image(void) {
            width = height = 0;
            data = NULL;
        }
        ~Image(void) {
            if (data != NULL)
                delete[] data;
        }
    
    int width;
    int height;
    Vec3f *data;
};
```

Constructors

Constructors can also take parameters

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

Using this constructor with stack or heap allocation:

```c
Image myImage = Image(10, 10); stack allocation
Image *imagePtr;
imagePtr = new Image(10, 10); heap allocation
```
### The Copy Constructor

```cpp
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:

```cpp
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.

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

**Computationally expensive**

It’s much faster to pass by reference:

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

or

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

### Class Hierarchy

Child classes can *call* parent functions

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

Call the parent constructor

Child classes can *override* parent functions

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

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

### Virtual Functions

A superclass pointer can reference a subclass object

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

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

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

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

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

### The main function

This is where your code begins execution

```cpp
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:

- Access outside of array bounds
  ```
  int intArray[10];
  intArray[10] = 6837;
  ```

- Attempt to access a NULL or previously deleted pointer
  ```
  Image *img;
  img->SetAllPixels(ClearColor);
  ```

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`
  ```
  Vec3f4 operator+(Vec3f4 a, Vec3f4 b);
  ```