

Lesson 5: Writing an Arrays



Writing data structures in C++

Compiled code

- C++ like C must be compiled and linked
- Compiling turns the code into machine code (*.o files) with calls to external libraries
- Linking actually links the external libraries with the code to produce an executable file
- We use a **Makefile** to do the compile and linking

```
all: main run

main: array.h array.cc main.cc
    g++ main.cc array.cc -o main

run: main
    ./main
```

array.h

```
#ifndef ARRAY_H
#define ARRAY_H

class Array {
private:
    int *data;
public:
    Array(int n);
    void set(int index, int value);
    int get(int index);
};

#endif
```

Operator Overloading

- Cpp is just ugly
- C++ allows us to overload operators (e.g. +, +=, <<, etc.)
- One operator is indexing: **operator[int]()**
- We can use this to return a reference to **data[i]**

```
int& Array::operator[](int index) {
    return data[index];
}
```

- This is not being taught as a lecture, but run as a practical session
- We are going to write a resizable array class in C++
- I will call the class Array although this is not a great name
- The point of this is to understand the subtleties of coding in C++

Cpp Style Classes: main.cc

```
#include <iostream>
#include "array.h"
using namespace std;

int main() {
    Array a(3);
    a.set(0,0);
    a.set(1,2);
    a.set(2,4);

    cout << a.get(0) << ", " << a.get(1) << ", " << a.get(2) << endl;

    return 0;
}
```

array.cc

```
#include "array.h"

Array::Array(int n) {
    data = new int[n];
}

void Array::set(int index, int value) {
    data[index] = value;
}

int Array::get(int index) {
    return data[index];
}
```

Updated main.cc

```
#include <iostream>
#include "array.h"
using namespace std;

int main() {
    Array a(3);

    for(int i=0; i<3; i++) {
        a[i] = i*i;
    }

    cout << a[0] << ", " << a[1] << ", " << a[2] << endl;

    return 0;
}
```

- As we might want to print different arrays lets create a print function
- We want Array to know how many elements are in it

```
#ifndef ARRAY_H
#define ARRAY_H

class Array {
private:
    int *data;
    int length;
public:
    Array(int n);
    int& operator[] (int index);
    int size();
};

#endif
```

```
#include <iostream>
#include "array.h"
using namespace std;

void print(Array& a, string name) {
    cout << name;
    for(int i=0; i<a.size(); i++) {
        cout << " " << a[i];
    }
    cout << endl;
}

int main() {
    Array a(10);

    for(int i=0; i<a.size(); i++) {
        a[i] = i*i;
    }

    print(a, "a:");

    return 0;
}
```

Copy Constructor

- C++ conveniently generates a copy constructor

```
Array b(a);
```
- Unfortunately this copies the address to `data` and the `length`
- But this is a *shallow copy* which means that both arrays work on the same data array
- This would be deeply confusing. Instead we have to write our own *copy constructor* to do a deep copy

```
Array::Array(Array& other) {
    data = new int[other.size()];
    length = other.size();
    for(int i=0; i<size(); ++i) {
        data[i] = other[i];
    }
}
```