C++ Quick Review

By Kamyar Allahverdi

A Sample Program

```
#include <iostream>
using namespace std;

int main() {
   cout << "hello, world!" << endl;
   return 0;
}
```

Variable Declaration

```
int i; // i is an integer
int & r; // r is a reference to an integer
int * p; // p is a pointer to an integer
int * & q; // q is a reference to a pointer to an integer
int * const c; // c is a const pointer to an integer
int const * d; // d is a pointer to a const integer
```

Heap vs. Stack

- Stack memory is managed.
  - Mostly keeps track of local variables
- Heap is a dynamic memory.
  - It’s a bigger memory, for big chunks of data

Heap in C++

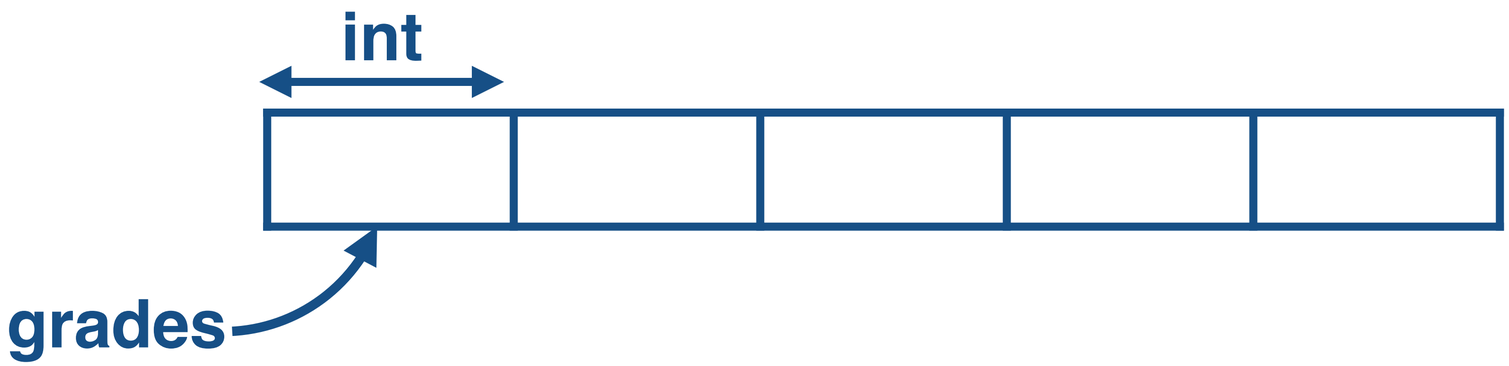
- C++ has two operators to allocate dynamic memory
  - **new/new[]**
  - **delete/delete[]**

Heap in C++

- An example:

```
int * grades;
grades = new int [5];
```

- Here grades will point to the starting point of a block of memory:



Heap in C++

- Problems with dynamic memory?
  - **Memory Leak:** If you lose track of memory allocated.
  - **Segmentation Fault**: If you access illegal memory address
- With C++11 and later, you should avoid new/delete entirely.

Standard Library

- We have a standard library in C++, that helps with managing memory.
- How?

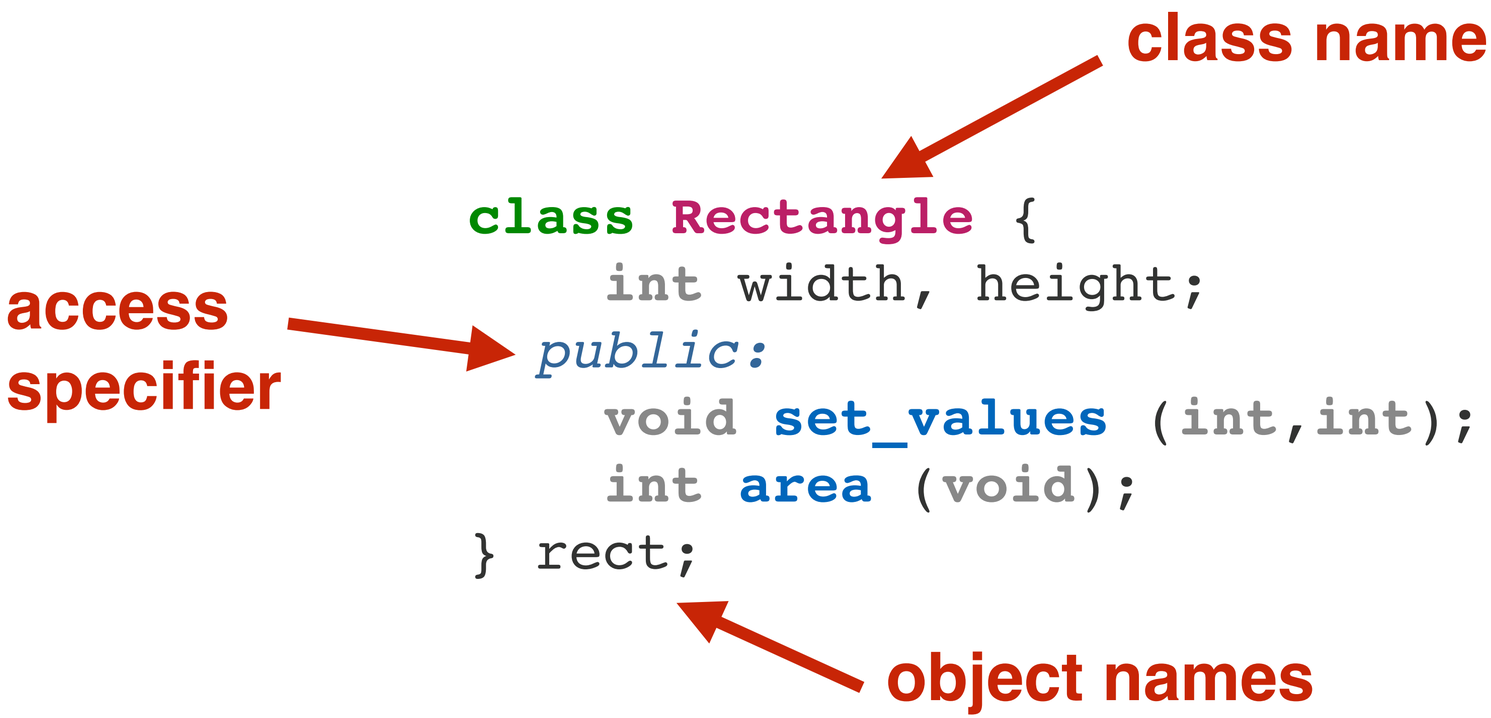
C++ Classes

- A simple class:

```
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area (void);
} rect;
```

C++ Classes

- A simple class:



C++ Classes

- Define members outside class declaration:

```
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
```

C++ Classes

- Memory is managed using Constructors and Destructors.
- When the class is deleted, the destructor is called and no leak happens.

```
class String
{
public:
String(int size)
    :str(NULL),
    size(size)
{
    str = new char[size];
}

~String() //destructor
{
    delete [] str;
};
private:
    char *str;
    int size;
}
```

namespace std

- Fast efficient containers.
- Most usable for us: **std::vector**

std::vector

```
int main()
{
    std::vector<std::string> words {"hello", "world!"};
    for(auto s: words)
        std::cout << s << ' ';
    std::cout << std::endl;
    return 0;
}
```

std::vector

- You can access the underlying array:
  - **myvector.data()**
- If you know the final size of the vector:
  - **std::vector myvector(1500);**

Macros

- They are replaced during compilation.

```
#include "myheader.hpp"
#define M_PI 3.14159
```

Header files

- Contains mostly definitions, and global variables
- You shouldn’t include them more than once
- We use macros to prevent this.

Header files

```
#ifndef MYHEADER_H
#define MYHEADER_H

int time_travel(int when);
// .. rest of header content

#endif // end of include guards
```