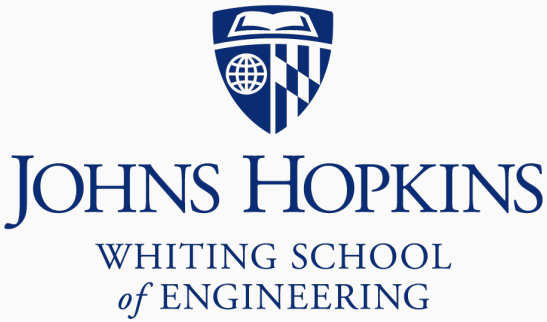
Constructors

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

Constructors

What are the fields of a class set to if we don't explicitly initialize them?

C++ classes

```
#include <vector>

class GradeList {
public:
    void add(double grade) {
        grades.push_back(grade);
        is_sorted = false;
    }

    double percentile(double percentile);
    double mean();
    double median();

private:
    std::vector<double> grades;
    bool is_sorted;
};
```

C++ classes

```
int main() {
    GradeList gl;
```

What are the values of `gl.grades` & `gl.is_sorted` right now?

I haven't set them to anything; are they set to reasonable defaults?

Not in general

- `gl.grades` *is* initialized properly via its *default constructor* (discussed soon)
- According to C++11 standard, `gl.is_sorted` is *uninitialized*!

Constructors

When we define a `class`, we can define one or more *constructors*

Each constructor is a way to build a valid instance of that `class`, with fields sensibly initialized

If you define no constructors, an *implicit default constructor* is automatically added by the compiler

Constructors

```
int main() {
    // behind the scenes, GradeList's implicit
    // default constructor is called...
    GradeList gl;

    // ...but it does not initialize gl.is_sorted!
    ...
}
```

We will prefer to define the constructor ourselves

Constructors

Terminology:

- *Default constructor* is a constructor that takes no arguments
  - We will see non-default constructors later
- *Implicit* default constructor is the default constructor that the compiler adds when no constructors are specified

Constructors

A constructor is a `public` member function with the same name as the `class`

```
class GradeList {
public:
    // default constructor for GradeList
    GradeList() { ... }
    ...
}
```

It has no return type; it doesn't return anything

Constructors

We've used default constructors already:

```
// vector<int>'s default constructor initializes empty vector
vector<int> my_ints;

// string's default constructor initializes empty string
string word;
```

Constructors

We cannot call a constructor directly. Constructor is called automatically when a new object is declared, or created using new

```
int main() {
    // calls default constructor for gl
    GradeList gl;

    // calls default constructor for *glp
    GradeList *glp = new GradeList();
}
```

(new discussed later)

Constructors

Constructors often use a special syntax called an *initializer list*:

```
class GradeList {
public:
    // Define our own "default constructor,"
    GradeList() : grades(), is_sorted(false) { }

    ...

private:
    std::vector<double> grades;
    bool is_sorted;
};
```

Constructors

```
class GradeList {
public:
    // Define our own "default constructor,"
    GradeList() : grades(), is_sorted(false) { }
    //            ^^^^^^^^  ^^^^^^^^^^^^^^^^
    //            Initializes grades by calling
    //            std::vector<int>'s default constructor
    //
    //            Initializes is_sorted by setting it to
    //            false

    ...

private:
    std::vector<double> grades;
    bool is_sorted;
};
```

Constructors

These default constructors have the same effect:

```
class IntAndString1 {
public:
    IntAndString1() : i(7), s("hello") { }
    //              ^^^^^^^^^^^^^^^^^^^^
    //               "initializer list"
    int i;
    std::string s;
};

class IntAndString2 {
public:
    IntAndString2() {
        i = 7;
        s = "hello";
    }
    int i;
    std::string s;
};
```

Constructors

```
#include <iostream>
#include "def_ctor_1.h"

using std::cout;  using std::endl;

int main() {
    IntAndString1 is1;
    IntAndString2 is2;
    cout << "is1.i=" << is1.i << ", is1.s=" << is1.s << endl;
    cout << "is2.i=" << is2.i << ", is2.s=" << is2.s << endl;
    return 0;
}
```

```
$ g++ -c def_ctor_1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o def_ctor_1 def_ctor_1.o
$ ./def_ctor_1
is1.i=7, is1.s=hello
is2.i=7, is2.s=hello
```

Constructors

Initializer list is the better choice and we will prefer it

- Works as expected both for normal and for reference variables
- Works both for using default and non-default constructors when initializing fields

```
IntAndString() : i(7), s("hello") { }

IntAndString() {
    i = 7;
    s = "hello";
}
```

Neither Java nor Python have initializer list syntax:

- stackoverflow.com/questions/7154654 in this course; it is clearer & less error-prone to initialize only in constructors