ECE198KL

L40PA
24 April 2013

access rights in C++
Constructors & destructors
new & delete

overloading & references

P13 up on Monday

Competition!

~~L39~~ L40P1

Types of access rights in C++

default → "private:"
- access allowed only within the class (class' functions)
- used for fields & implementation functions ~~static~~ class variables,

"protected:"
- access allowed within the class and any derived classes
- use similar to private

"public:"
- access allowed to any code
- used for interface functions, and for instance initialization/teardown

---

Access is specified for all subsequent declarations (but can be changed by another specifier):

```
class ALPHA {
    private:
        ...   ] all private
    public:
        ...   ] all public
    private:
        ...   ] all private
};
```

L39PG
L40P2

Constructors

A constructor is a subroutine called to initialize a new variable of class type (an instance of the class — often called an object).

Constructor is always called for class instances:

static variable — before main

automatic variable — start of block of code / definition point in code

dynamic variable — just after allocation

A Few Important Details...

- "Name" of constructor is name of class
- No return type
- Can have >1 if compiler can tell which one you want to use (more later, but basically args must match)

- Two created by default
  - no args: used for array element initialization; if you define any other constructor, this variant is not created, and arrays can only be declared if you define a constructor with no args
  - copy constructor: takes one argument: another instance of class

```
ALPHA a;
ALPHA b=a;   <- used here
```

more detail later

L39 p67
L40 p3

```
class ALPHA {
    private:
        int one;
        int two;
    public:
        ALPHA (int arg, int second);
};
```

In source file, the definition:

```
ALPHA::ALPHA (int arg, int second) : one(arg), two(second)
{
    // code
}
```

base class constructor would go here

optional list of initializers:
field (expression)

constructor order of execution
- base class constructor (if only) can be chosen as initializer
- initializers in order of field declaration
  (how you list them in constructor does not affect code — put them in same order)
  NOTE: constructors called for all class type fields, even if you give no initializer.
- body of constructor

L39P8
L40P4

destructors

A destructor is a subroutine called to destroy a variable of a class type (an object)

Destructor is always* called for objects:

static variable: after main

automatic variable: just before RET instruction

dynamic variable: just before free [delete in C++]

*- possibly skipped if program crashes

[- supposed to execute if exception thrown from child to parent... may depend on use of one compiler for all code]

A Few Important Details...

Define one destructor, and make it virtual

```
virtual ~ALPHA();   // no args to destructor
```

Why virtual? Avoid calling only base class' destructor... (for dynamic variables)

Order of execution

- body of destructor
- destructors for fields with class type (bottom to top of order in class definition)
- base class destructor (if any)

NOTE: A pointer is not an object. If you have a pointer to a private, dynamically allocated object, your destructor must explicitly delete that object.

allocating memory

malloc & free do not call constructors/destructors

⇒ do not use for class instances
(in new C++ code, best just not to use them)

create a new object:

```
Class* ptr = new Class (arg1, arg2, ...);
```

(arg1, arg2, ...) — passed to Class' constructor (omitted ⇒ no arguments)

returns pointer to constructed Class instance (Class*)

On failure, throws an exception. We probably won't get to exceptions in our class, but by default kills your program.

If you want call to return NULL instead (no constructor called in that case),

```
#include <new>
Class* ptr = new (std::nothrow) Class (arg1, arg2, ...);
```

If you want to allocate an array, use---

```
Class *ptr = new Class[42];
```

↑ array of 42 objects of type Class, each constructed using constructor with no args (which must be defined)

add "(std::nothrow)" & #include <new> at top of file if you want new to return NULL on failure

Deleting objects

You (the programmer) must remember whether a pointer points to an object or an array.

Call the right form of delete!
(Bug likely to be hard to find if you do not.)

```
Class* ptr;
delete ptr;    // object — no args allowed to destructor
delete[] ptr;  // array — also no args allowed
```

It's ok to delete NULL (has no effect) in C++