Poll Everywhere

pollev.com/cse333

- How is Homework 2 coming along?

A. Made some progress
B. Haven't started yet
C. I'm stuck trying to solve part of it
D. I'm done

C++ Constructor Insanity (cont'd)

CSE 333 Autumn 2021

Instructor: Chris Thachuk

Teaching Assistants:

Arpad (John) Depaszthory
Ian Hsiao
Logan Gnanapragasam
Mengqi (Frank) Chen
Angela Xu
Khang Vinh Phan
Maruchi Kim
Cosmo Wang

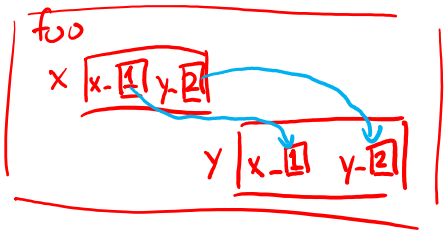
Administrivia

- Exercise 9 out, due Wed (Oct. 27) @ 10am
  - Operators will be covered in Monday's lecture
- Homework 2 due next Thursday (Oct. 28)
- Midterm exam on Friday, Nov. 5
  - Take-home (completed independently)
  - Exam will be open for 24 hours on Gradescope
  - Can start / stop / resume all day
  - More details will be posted on class website

Lecture Outline

- Constructors
- **Copy Constructors (restart)**
- Assignment
- Destructors

Copy Constructors



- C++ has the notion of a copy constructor (cctor)
  - Used to create a new object as a copy of an existing object

```
Point::Point(const int x, const int y) : x_(x), y_(y) { }

// copy constructor
Point::Point(const Point& copyme) {
  x_ = copyme.x_;
  y_ = copyme.y_;
}

void foo() {
  Point x(1, 2);  // invokes the 2-int-arguments constructor

  Point y(x);     // invokes the copy constructor
                  // could also be written as "Point y = x;"
}
```

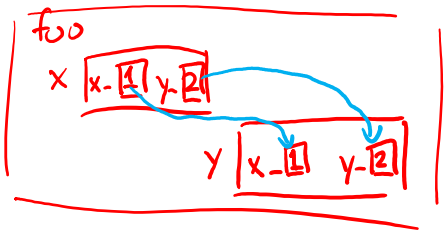
reference to object of same class

alias binds to object

constructing from existing object, so we use the copy ctor.

a ctor must be called because the object didn't exist previously.

Copy Constructors



- C++ has the notion of a copy constructor (cctor)
  - Used to create a new object as a copy of an existing object

```
Point::Point(const int x, const int y) : x_(x), y_(y) { }

// copy constructor
Point::Point(const Point& copyme) {
  x_ = copyme.x_;
  y_ = copyme.y_;
}

void foo() {
  Point x(1, 2);  // invokes the 2-int-arguments constructor

  Point y(x);     // invokes the copy constructor
                  // could also be written as "Point y = x;"
}
```

reference to object of same class

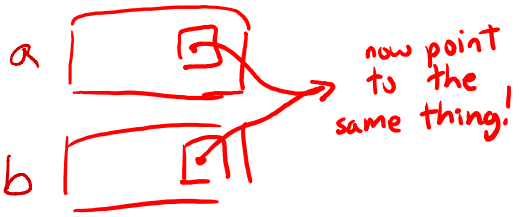
alias binds to object

constructing from existing object, so we use the copy ctor.

a ctor must be called because the object didn't exist previously.

  - Initializer lists can also be used in copy constructors (preferred)

Synthesized Copy Constructor



- If you don't define your own copy constructor, C++ will synthesize one for you
  - It will do a *shallow* copy of all of the fields (*i.e.*, member variables) of your class (can be problematic with pointers)
  - Sometimes the right thing; sometimes the wrong thing

```
#include "SimplePoint.h"

... // definitions for Distance() and SetLocation()

int main(int argc, char** argv) {
  SimplePoint x;
  SimplePoint y(x);  // invokes synthesized copy constructor
  ...
  return EXIT_SUCCESS;
}
```

When Do Copies Happen?

- The copy constructor is invoked if:

When Do Copies Happen?

- The copy constructor is invoked if:
  - You *initialize* an object from another object of the same type:

```
Point x;        // default ctor
Point y(x);     // copy ctor
Point z = y;    // copy ctor
```

When Do Copies Happen?

- The copy constructor is invoked if:
  - You *initialize* an object from another object of the same type:

```
Point x;        // default ctor
Point y(x);     // copy ctor
Point z = y;    // copy ctor
```

  - You pass a non-reference object as a value parameter to a function:

```
void foo(Point x) { ... }

Point y;        // default ctor
foo(y);         // copy ctor
```

I

When Do Copies Happen?

- The copy constructor is invoked if:
  - You *initialize* an object from another object of the same type:

```
Point x;        // default ctor
Point y(x);     // copy ctor
Point z = y;    // copy ctor
```

  - You pass a non-reference object as a value parameter to a function:

```
void foo(Point x) { ... }

Point y;        // default ctor
foo(y);         // copy ctor
```

pass by value

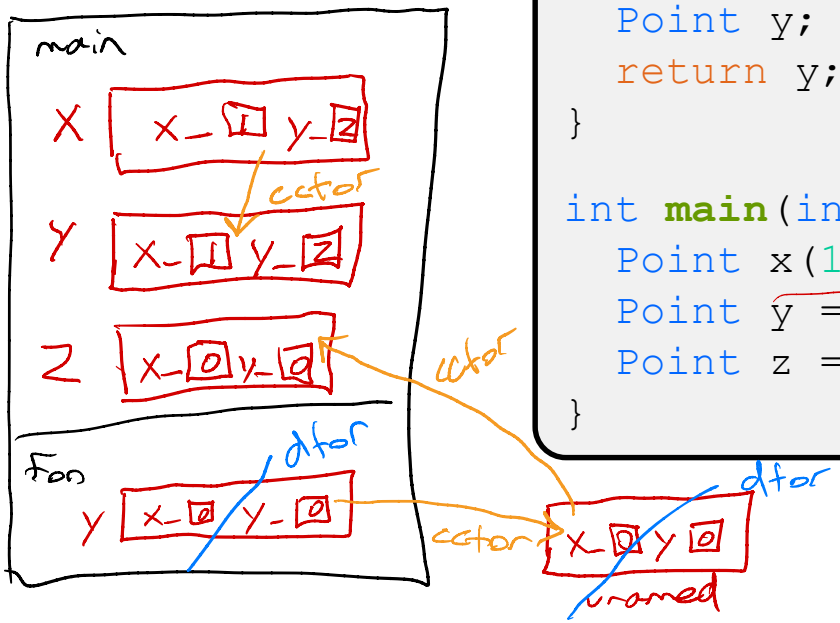
  - You return a non-reference object value from a function:

```
Point foo() {
  Point y;      // default ctor
  return y;     // copy ctor
}
```

1

Compiler Optimization

- The compiler sometimes uses a "return by value optimization" or "move semantics" to eliminate unnecessary copies
  - Sometimes you might not see a constructor get invoked when you might expect it




```
Point foo() {
  Point y;          // default ctor
  return y;         // copy ctor? optimized?
}

int main(int argc, char** argv) {
  Point x(1, 2);    // two-ints-argument ctor
  Point y = x;      // copy ctor
  Point z = foo();  // copy ctor? optimized?
}
```

Lecture Outline

- Constructors
- Copy Constructors
- **Assignment**
- Destructors

I

Assignment != Construction

- "=" is the assignment operator
  - Assigns values to an *existing, already constructed* object

```
Point w;          // default ctor
Point x(1, 2);    // two-ints-argument ctor
Point y(x);       // copy ctor
Point z = w;      // copy ctor
y = x;            // assignment operator
```

z did not exist

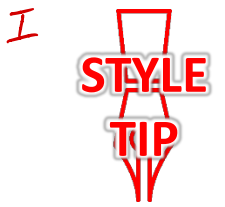
y already existed

Overloading the "=" Operator

- You can choose to define the "=" operator
  - But there are some rules you should follow:

```
Point& Point::operator=(const Point& rhs) {
  if (this != &rhs) {  // (1) always check against this
    x_ = rhs.x_;
    y_ = rhs.y_;
  }
  return *this;        // (2) always return *this from op=
}
```

Overloading the "=" Operator



- You can choose to define the "=" operator
  - But there are some rules you should follow:

```
Point& Point::operator=(const Point& rhs) {
  if (this != &rhs) {  // (1) always check against this
    x_ = rhs.x_;
    y_ = rhs.y_;
  }
  return *this;        // (2) always return *this from op=
}

Point a;          // default constructor
a = b = c;        // works because = return *this
a = (b = c);      // equiv. to above (= is right-associative)
(a = b) = c;      // "works" because = returns a non-const
```

more important when there is dynamically allocated memory.

a.operator=(b.operator=(c))

I

Synthesized Assignment Operator

- If you don't define the assignment operator, C++ will synthesize one for you
  - It will do a *shallow* copy of all of the fields (*i.e.*, member variables) of your class
  - Sometimes the right thing; sometimes the wrong thing

issues with dynamic memory

```
#include "SimplePoint.h"

... // definitions for Distance() and SetLocation()

int main(int argc, char** argv) {
  SimplePoint x;
  SimplePoint y(x);
  y = x;              // invokes synthesized assignment operator
  return EXIT_SUCCESS;
}
```

Lecture Outline

- Constructors
- Copy Constructors
- Assignment
- **Destructors**

I

Destructors

- C++ has the notion of a destructor (dtor)
  - Invoked automatically when a class instance is deleted, goes out of scope, etc. (even via exceptions or other causes!)
  - Place to put your cleanup code – free any dynamic storage or other resources owned by the object
  - Standard C++ idiom for managing dynamic resources
    - Slogan: "*Resource Acquisition Is Initialization*" (RAII)

tilde

no parameters

```
Point::~Point() {   // destructor
  // do any cleanup needed when a Point object goes away
  // (nothing to do here since we have no dynamic resources)
}
```

reverse of ctor: ① body of dtor

② destruct members in reverse order of declaration

I

Destructor Example

```
class FileDescriptor {
 public:
  FileDescriptor(char* file) {          // Constructor
    fd_ = open(file, O_RDONLY);
    // Error checking omitted
  }
  ~FileDescriptor() { close(fd_); }     // Destructor
  int get_fd() const { return fd_; }    // inline member function
 private:
  int fd_;  // data member
};  // class FileDescriptor
```

automatically close file on destruction

FileDescriptor.h

```
#include "FileDescriptor.h"

int main(int argc, char** argv) {
  FileDescriptor fd(foo.txt);
  return EXIT_SUCCESS;
}
```

destructor is called

Poll Everywhere

pollev.com/cse333

- How many times does the ***destructor*** get invoked?
  - Assume `Point` with everything defined (ctor, cctor, =, dtor)
  - Assume no compiler optimizations

test.cc

```
Point PrintRad(Point& pt) {
  Point origin(0, 0);
  double r = origin.Distance(pt);
  double theta = atan2(pt.get_y(), pt.get_x());
  cout << "r = " << r << endl;
  cout << "theta = " << theta << " rad" << endl;
  return pt;
}

int main(int argc, char** argv) {
  Point pt(3, 4);
  PrintRad(pt);
  return EXIT_SUCCESS;
}
```

A. 1
B. 2
C. 3
D. 4
E. We're lost…

Class Definition (from previous lecture)

Point.h

```
#ifndef POINT_H_
#define POINT_H_

class Point {
 public:
  Point(int x, int y);                    // constructor
  int get_x() const { return x_; }        // inline member function
  int get_y() const { return y_; }        // inline member function
  double Distance(const Point& p) const;  // member function
  void SetLocation(int x, int y);         // member function

 private:
  int x_;  // data member
  int y_;  // data member
};  // class Point

#endif  // POINT_H_
```

this const means that this function is not allowed to change the object on which it is called (the implicit "this" pointer)

function definitions

declarations

compiler may choose to expand inline (like a macro) instead on an actual function call

naming convention for class data members (Google C++ style guide)

Poll Everywhere

pollev.com/cse333

I

- How many times does the ***destructor*** get invoked?

ctor	cctor	op=	dtor
2	1	0	3

test.cc

```
Point PrintRad(Point& pt) {
  Point origin(0, 0); // (2) ctor called
  double r = origin.Distance(pt); // takes a ref, pt is NOT copied
  double theta = atan2(pt.get_y(), pt.get_x());
  cout << "r = " << r << endl;
  cout << "theta = " << theta << " rad" << endl;
  return pt; (3) return an object, so cctor is called to create a temp
}            (4) during cleanup, origin is destructed

int main(int argc, char** argv) {
  Point pt(3, 4); // (1) ctor called
  PrintRad(pt);  // takes a ref, so pt is NOT copied
  return EXIT_SUCCESS; (5) return of PrintRad ignored; temp is destructed
}                      (6) during cleanup, pt is destructed
```

Extra Exercise #1

- Modify your Point3D class from Lec 9 Extra #1
  - Disable the copy constructor and assignment operator
  - Attempt to use copy & assignment in code and see what error the compiler generates
  - Write a `CopyFrom()` member function and try using it instead
    - (See details about `CopyFrom()` in next lecture)

Extra Exercise #2

- Write a C++ class that:
  - Is given the name of a file as a constructor argument
  - Has a `GetNextWord()` method that returns the next whitespace- or newline-separated word from the file as a copy of a `string` object, or an empty string once you hit EOF
  - Has a destructor that cleans up anything that needs cleaning up