About how long did Exercise 5 take you?

A. [0, 2) hours
B. [2, 4) hours
C. [4, 6) hours
D. [6, 8) hours
E. 8+ Hours
F. I didn’t submit / I prefer not to say
C++ Constructor Insanity (cont’d)
CSE 333 Fall 2023

Instructor: Chris Thachuk

Teaching Assistants:

Ann Baturytski  Humza Lala
Yuquan Deng  Alan Li
Noa Ferman  Leanna Mi Nguyen
James Froelich  Chanh Truong
Hannah Jiang  Jennifer Xu
Yegor Kuznetsov
Relevant Course Information

- Exercise 6 released yesterday, due next Monday Wednesday (10/25)
  - Write a substantive class in C++

- Midterm in next Friday’s class (10/27)
  - See course website for details & sample midterms
  - Review session will go forward Monday evening (zoom); see Ed post tomorrow with details

- Homework 2 due on 10/30
  - See Ed post about partner finding & confirmation
Lecture Outline

- Constructors (*covered last lecture*)
- Copy Constructors (*covered last lecture*)
- Assignment
- Destructors
Assignment != Construction

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

```c++
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
```
Overloading the “=” Operator

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

```cpp
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
```
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

```
#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
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)

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

```cpp
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

#include "FileDescriptor.h"

int main(int argc, char** argv) {
    FileDescriptor fd("foo.txt");
    return EXIT_SUCCESS;
}
```
How many times does the **destructor** get invoked?

- Assume **Point** with everything defined (ctor,cctor, =, dtor)
- Assume no compiler optimizations

---

A. 1
B. 2
C. 3
D. 4
E. We’re lost…

```cpp
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;
}
```
Class Definition (from last lecture)

```cpp
#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_
```

- `#ifndef POINT_H_` and `#define POINT_H_` are used to prevent redefinition.
- `public` and `private` sections define access levels for members.
- Constructor: `Point(int x, int y);` is declared in the public section.
- Inline member functions: `int get_x() const` and `int get_y() const` are defined inline within the class definition.
- Member functions: `double Distance(const Point& p) const` and `void SetLocation(int x, int y);` are defined in the public section.
- Data members: `int x_;` and `int y_;` are declared in the private section.

This setup allows for encapsulation (data hiding) and provides a clear separation between public and private members.

- Use of `const` for member functions allows the function to operate on a `const` object without modifying it.
- The naming convention for class data members is `int x_;` and `int y_;` (Google C++ style guide).

This class structure is fundamental in object-oriented programming, providing a blueprint for creating objects with specific behaviors and properties.
How many times does the destructor get invoked?

<table>
<thead>
<tr>
<th>ctor</th>
<th>cctor</th>
<th>op=</th>
<th>dtor</th>
</tr>
</thead>
</table>

```cpp
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;
}
```
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

#include "FileDescriptor.h"

int main(int argc, char** argv) {
    FileDescriptor fd1(foo.txt);
    File Descriptor fd2(fd); // Invokes synthesized cctor
    return EXIT_SUCCESS; // What happens when we return and destruct our objects?
}

(This won’t crash the program, but what if we were using heap allocation instead of file descriptors?)
Extra Exercise #1

- Write a C++ program that:
  - Has a class representing a 3-dimensional point
  - Has the following methods:
    - Return the inner product of two 3D points
    - Return the distance between two 3D points
    - Accessors and mutators for the $x$, $y$, and $z$ coordinates
Extra Exercise #2

Write a C++ program that:

- Has a class representing a 3-dimensional box
  - Use your Extra Exercise #1 class to store the coordinates of the vertices that define the box
  - Assume the box has right-angles only and its faces are parallel to the axes, so you only need 2 vertices to define it

- Has the following methods:
  - Test if one box is inside another box
  - Return the volume of a box
  - Handles <<, =, and a copy constructor
  - Uses const in all the right places
Extra Exercise #3

- Modify your Point3D class from Extra Exercise #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 #4

- 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