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
Yuquan Deng
Noa Ferman
James Froelich
Hannah Jiang
Yegor Kuznetsov

Humza Lala
Alan Li
Leanna Mi Nguyen
Chanh Truong
Jennifer Xu
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

```cpp
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

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

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;
}

/* Destructor automatically closes file for the user! */
/* destruct object when it falls out of scope (here, when we return) */
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…
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_
```

- 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
- Naming convention for class data members (Google C++ style guide)
- Compiler may choose to expand inline (like a macro) instead of an actual function call.
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>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

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

int main(int argc, char** argv) {
    Point pt(3, 4);   // 4 ctor called
    PrintRad(pt);   // PrintRad takes ref, so pt is NOT copied
    return EXIT_SUCCESS;   // 5 return value of PrintRad ignored; temp is destructed
}    // 6 while cleaning up, pt is destructed
```
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);
  FileDescriptor fd2(fd_); // Invokes synthesized cctor
  return EXIT_SUCCESS;
}
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