CS 162
Intro to Computer Science II

Lecture 14
Shallow vs. deep copy
Big 3
2/19/24
Odds and Ends

• Sign up for assignment 2 demo ASAP!

• Lab 7 posted

• Assignment 3 rubrics posted

• Design 3 (ex. + doc) + Quiz 3 past due
Today’s topics

• Midterm Report
• Shallow vs. Deep Copy
• Big 3 Implementation
Shallow Copy vs. Deep Copy

• Shallow:
  - A.k.a.: member-wise copy
  - Copy the contents of member variables from one object to another
  - Default behavior when objects are copied or assigned

```
Objetct1
  var1
  var2
  var3

Objetct2
  var1
  var2
  var3
```
Shallow Copy vs. Deep Copy

• Shallow:
  • What if the object has dynamic memory allocated?

  • This could be problematic as if we make any changes to the array in object 1, object 2 will be affected as well...
Shallow Copy vs. Deep Copy

• Deep:
  • Copy what each member variable is pointing to so that you get a separate but identical copy
  • Has to be programmer-specified
Assignment Operator (=) Overload

• Predefined assignment operator returns a reference
  • Allows us to chain assignments together: \( a = b = c \)
    • First set “\( b = c \)” and return a reference to \( b \). Then set “\( a = b \)”
    • Need to make sure the assignment operator returns something of the same type as its left hand side

• Overloading assignment operator
  • Must be a member of the class
Assignment Operator (\(=\)) Overload

- Ex:

  ```cpp
  Course& Course::operator=(const Course& obj) {
    //pay attention to the return type
    this->title = obj.title; //for non-dyn. Memory, shallow copy
    this->enroll = obj.enroll;
    this->instructor = obj.instructor;
    if (this->roster != nullptr) //if the ptr has memory allocated
      delete [] this->roster; //free it
    this->roster = new string [this->enroll]; //deep copy
    for (int i = 0; i < this->enroll; i++)
      this->roster[i] = obj.roster[i];
    return *this; //return the calling obj
  }
  ```
Copy Constructor

• Constructor that has one parameter that is of the same type as the class
  • Has to accept reference as parameter (normally `const`)
  • Allows for distinct copies, changes to one does not impact the other
  • Called automatically in three cases:
    • When a class object is being declared and initialized by another object of same type
    • Whenever an argument of the class type is “plugged in” for a call by value parameter
    • When a function returns a value of the class type
Copy Constructor

Ex:

Course::Course(const Course& obj) {
    this->title = obj.title; //pay attention to the parameter
    this->enroll = obj.enroll; //for non-dyn. Memory, shallow copy
    this->instructor = obj.instructor;
    this->roster = new string[this->enroll]; //deep copy
    for (int i = 0; i < this->enroll; i++)
        this->roster[i] = obj.roster[i]; //no return

}
Destructor

• Delete the object
• Will be automatically created if one is not supplied
  • Will not handle dynamic memory
• ~Class_name(); // no return type, no parameters, only one allowed

• Called when the object goes out of scope
  • When the function ends
  • When the program ends
  • A block containing local variables ends
  • A delete operator is called
Destructors

Ex.:

```cpp
Course::~Course() {
    if (this->roster != nullptr) {
        //if the ptr has memory allocated
        delete [] this->roster;       //free it
        this->roster = nullptr;
    }
}
```
The Big Three

• If you implement either a **Destructor**, a **Copy Constructor**, or an **Overloaded Assignment Operator**, you should ensure that all 3 are defined

• If you needed one, you probably need all of them

• This rule of thumb goes by several names:
  • The Big Three
  • The Rule of Three
  • The Law of The Big Three

• *C++11 has an expanded version: The Big 5
  • We won’t cover this yet
# Big Three Activity

<table>
<thead>
<tr>
<th>Function</th>
<th>Prototype</th>
<th>Job</th>
<th>When is it called</th>
<th>Default Behavior if not defined?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor</td>
<td>ClassName(); ClassName(w/ params)</td>
<td>Build the object</td>
<td>Default is called when object is declared with no parameters and no “=“ sign. Nondefault is called if parameters are given</td>
<td>The compiler will provide a default one. It will initialize all variables with garbage values, will not set up pointers</td>
</tr>
<tr>
<td>Copy Constructor</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment Operator Overload</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destructor</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Asm3 Hints:

• Which class needs Big 3?
• Where to implement the “add a flight” functionality?
• Where to implement the “remove a flight” functionality?

• Is it a good practice to access Flight internals from the Manager class?
  • i.e., get_airports()[0].get_flight()[0].get_flight_number()?
  • NO!!! THIS VIOLATES THE RULE OF ENCAPSULATION!!!!

• Game flow?
• What’s inside your main()? driver.cpp?
• Frequently check memory leaks!!!