### 11: **Destructor**

### **Purpose** - it cleans up all resources held by the class or objects through cleaning up heap memory and closing all the files

* + If we ever used **new** keyword, we have to free the memory (calling **delete**) so that we don’t leak memory.

#### **Automatic Destructor**

* + It exists only if no custom destructor is defined
  + **Functionality** - It only calls the destructor of the members without doing anything else ie.cleaning heap memory or closing any files
  + **Invoked** - it is always automatically called when reclaimed
    - Stack memory: reclaimed when function returns
    - Heap memory: reclaimed when calling delete
  + Destructor is the final thing to call in the lifecycle of a class.

#### **Custom destructor**

* + If our member variables are on heap, we need to define a custom destructor
  + It does not take any parameters ie ~cube()
  + It does not delete anything for you but let programmer in control of the timing of delete and ways of deleting by writing code - see example below

|  |  |  |  |
| --- | --- | --- | --- |
| cube.h | | cube.cpp | |
| 1  2  3  4  5  6  7  8  9  10  11  12  1314  15  16  17 | #pragma once  namespace cs225 {  class Cube {  public:  Cube(); Cube(double length);  Cube(const Cube & other);  ~Cube();  private:  double length\_;  };  } | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | namespace cs225 {  Cube::Cube() {  length\_ = 1;  }  Cube::Cube(double length) {  length\_ = length;  }  Cube::Cube(const Cube & other) {  length\_ = other.length\_;  }  Cube::~Cube() {  …..//define your destructor HERE  }  } |

#### **Overload operators in cpp**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cube.h |  | Cube.cpp |
| 1  2  3 | Cube operator+(const Cube & other) const;  Cube & operator=(const Cube & other); | 1  2  3  4  5  6  7  8  9  10 | Cube Cube::operator+(const Cube & other) const {  return joinCube(\*this, other);  }  Cube & Cube::operator=(const Cube & other) {  length\_ = other.length\_;  return;  } |

* + **Defining “+” op**
  + How to define the “+” operator in “cube c3 = c1 + c2”?
  + Define name of op, return type of op, the argument type we take into to be RHS. In this example function name is operator
  + Return type needs to be Cube instead of Cube & since reference variable does not take any space but we want to return an object
  + LHS is the instance + op is being called on so we use ‘\*this“ - a pointer to the instance of this class
  + **Assignment “=” op**
    - If we don’t define one, it provides auto free in every class
    - Similar to copy constructor
      * cube c2(c1)
    - Differ from copy constructor:
      * cube c1 c2;
      * c2 = c1;
    - Copy constructor: copies an object but it does not destroys an object
      * \_copy(other);
    - Destructor: does not copy object but it destroys
      * \_destroy();
    - Assignment op : it copies an object but also destroys itself
      * \_destroy(); // Clear the memory of the object.
      * \_copy(other); //Copy the state of the assigned object.

#### **Rule of Three**

* + If you define any one of these three functions, you should define **ALL** of them.
    - Assignment op
    - Copy constructor
    - destructor

#### **Inheritance:**

* + Classes can be extended to build other classes. We call the class being extended the base class and the class inheriting the functionality the derived class.
  + In the below example, class square inherit from class shape
    - Everything under **Shape public** is now in **Square public**
    - Do not get private variables and functions

|  |  |  |  |
| --- | --- | --- | --- |
| square.h | | square.cpp | |
| 1  2  3  4  5  6  7  8  9  10  11 | #pragma once  #include "Shape.h"  class Square : public Shape { //syntax of inherit  public:  Square();  Square(double length);  double getArea() const;  private:  }; | 1  2  3  4  5  6  7  8  9  10  11 | Square::Square() { }  Square::Square(double length) : Shape(length) { }  double Square::getArea() const {  return getLength() \* getLength();  } |