CSCI 335 Software Design and Analysis III Lecture 2: C++11

Professor Anita Raja

1

Agenda

- C++
  - Lvalues and Rvalues
  - Parameter Passing
  - Return Passing
  - std::swap and std::move
  - Big Five

2

"The Big Five!" (not Three)

- Destructor
- Copy Constructor
- Copy Assignment operator =
- Move Constructor
- Move Assignment operator =
- When do defaults fail?
  - Shallow copy vs. deep copy

3

When do defaults fail?

- Class that contains data member that is a pointer
  - Default destructor does nothing: do delete ourselves.
  - copy constructor and copy assignment operator will make .copies of pointer rather that objects that are being pointed at.
  - 2 class instances pointing to same object (called shallow copy),
- Want deep copy
  - Clone of entire object is made.
  - Implement destructor, copy assignment and copy constructor operators ourselves.
  - =>move constructor and move assignment must be implemented as well.
- Defaults use:
  - All or nothing

4

IntCell signature operations

```
~IntCell(); //Destructor
IntCell(const IntCell &rhs);  //Copy constructor
IntCell(IntCell && rhs);    //Move constructor
IntCell & operator= (const IntCell &rhs); //Copy assignment
IntCell & operator= (IntCell && rhs); //Move assignment
```

Operator= reference to the invoking object; to allow chained assignments a=b=c

stmt = default; //to allow default

stmt = delete; //to disallow

5

Modified IntCell to hold a pointer to an integer *sans "The Big Five"*

```
class IntCell {
  public:
    explicit IntCell(int initial_value = 0)
      { stored_value_ = new int{initial_value}; }
    int Read() const
      { return *stored_value_; }
    void Write( int x )
      { *stored_value_ = x; }
  private:
    int *stored_value_;
};

    int TestFunction() {
        IntCell a{2};
        IntCell b = a;
        IntCell c;
        c = b;
        a.Write(4);
        cout << a.Read() << endl << b.Read( ) << endl << c.Read( )
    << endl;
        return 0;
```

7

Correct Implementation with "The Big Five"

- If default does not make sense,
  - Implement destructor, copy-and-move constructors, and copy-and-move assignment operators.
- Copy assignment operator can be implemented
  - by creating a copy using the copy constructor
  - then swap it with the existing object.
- Move assignment operator implemented
  - swapping member by member.

8

Correct Implementation with "The Big Five" --

```
// Destructor
IntCell::~IntCell( )
{
    …
}
// Copy constructor
IntCell::IntCell(const IntCell & rhs)
{
    …
}
// Copy assignment operator
IntCell & IntCell::operator=(const IntCell & rhs)
{
    …
}
```

9

Correct Implementation with "The Big Five"

```
// Destructor - stops shallow copying
IntCell::~IntCell( )
{
    delete stored_value_;
}
// Copy constructor
IntCell::IntCell(const IntCell & rhs)
{
    stored_value_ = new int{*rhs.stored_value_};
}
// Copy assignment operator - check for aliasing
IntCell & IntCell::operator=(const IntCell & rhs)
{
    if (this != &rhs)
        *stored_value_ = *rhs.stored_value_; // assumes
initial value
    return *this;
}
```

10

C++11 Style Copy Assignment

```
// Copy assignment operator - check for aliasing
IntCell & IntCell::operator=(const IntCell & rhs)
{
    if (this != &rhs)
        *stored_value_ = *rhs.stored_value_; // assumes initial
value
    return *this;
}
```

- Uses standard idiom to check for aliasing
  - `if (this != &rhs)`
  - A self-assignment in which the client is making a call obj=obj.
- then copying each data field in turn as needed.
- On completiong, returns a reference to itself using *this.
- In C++11 copy assignment often written using a copy-and-swap idiom.

11

C++11 Style Copy Assignment

```
// Copy-and-swap idiom.
// In C++11 this is the usual implementation
IntCell & operator=( const IntCell &rhs ) {
      IntCell copy = rhs;  // Calls the copy-constructor
      std::swap(*this, copy);
      return *this;
  }
```

1. Place a copy of rhs into copy using the copy constructor.
2. copy is swapped into *this, placing the old contents into copy
3. In return, a destructor is invoked for copy, cleaning up the old memory.

Bit inefficient for intCell but good for others.

12

C++11 Style Copy Assignment

IMPORTANT NOTE:

- If swap is implemented using copy assignments,

```
void swap(double &x, double &y) {
    double tmp = x;
    x = y;
    y = tmp;
}
```

Copy-and-swap idiom will not work; there will be a mutual **non-terminating recursion**.

=> swap should be implemented either with three moves or swapping data member by data member.

13

Move Constructor and Move Assignment

```
// Move constructor
IntCell(IntCell && rhs) {
    …
}

// Move assignment operator
IntCell & operator=( IntCell && rhs ) {
    …
}
```

14

Move Constructor and Move Assignment

```
// Move constructor
IntCell(IntCell && rhs) : stored_value_{rhs.stored_value_}{
rhs.stored_value_ = nullptr; }


// Move assignment operator
IntCell & operator=(IntCell && rhs) {
    …
}
```

15

Move Constructor and Move Assignment

```
// Move constructor
IntCell(IntCell && rhs) : stored_value_{rhs.stored_value_}{
rhs.stored_value_ = nullptr; }


// Move assignment operator
IntCell & operator=(IntCell &&rhs) {
    // Use std::swap for all data members
    std::swap(stored_value_, rhs.stored_value_ );
    return *this;
}
```

16

Move Constructor and Move Assignment

```
// Expand IntCell so that it contains a vector:
//   private: vector<int> items_; //i.e. non-primitive type

// Move constructor
IntCell(IntCell && rhs) : stored_value_{rhs.stored_value_},
                          items_{std::move(rhs.items_) }
{ rhs.stored_value_ = nullptr; }


// Move assignment operator
IntCell & operator=( IntCell && rhs ) {
    // Use std::swap for all data members
    std::swap(stored_value_, rhs.stored_value_);
    std::swap(items_, rhs.items_);
    return *this;
}
```

17

"The Big Five! – Final notes"

- Default behavior can be stated:
  - `IntCell(const IntCell &rhs) = default;`
- Or the function can be disabled:
  - `IntCell(const IntCell &rhs) = delete;`
- Normally, if copy-constructor is disabled, then assignment operator should also be disabled:

```
 IntCell(const IntCell &rhs) = delete;
IntCell &operator=(const IntCell &rhs) = delete;
// If the above are deleted then, the expressions such as
// IntCell A = B; IntCell A{C}; … cause error.
```

- **If you implement one of the "big five", then you should implement all.**

18

Summary

- Big Five
- Parameter and Return Passing
- Next Class:
  - Templates
  - Matrices
  - Algorithmic Analysis
- HW1 Assigned Due 11:30pm Sep 15, 2022
- Next class:
  - Read Chapter 2 for next class
  - Work out these problems on your own:
    - What's 1+2+…+n? Prove it.
    - What's 1+2+4+…+2^n? Prove it.
    - If A_0=1 and A_n=2A_{n-1}+1, what's a closed form formula for A_n? Prove it.

19