Section 1.5
Constructors and Destructors

1. Default arguments
2. Default constructors
3. Destructors
4. Order of invocation
5. Copy constructors
6. Conversion constructors

1.5.1 Default Arguments

- Terminology
  - a function *argument* is the same as a parameter value
  - a *default* argument is a default parameter value
- Characteristics of default parameter values:
  - they are specified in the function prototype
  - one or more parameters may have default values
  - defaulted parameters must be right-most in the parameter list

Default Arguments (cont.)

- Uses of default arguments:
  - in global functions
  - in member functions
  - to combine a default constructor with multiple-parameter one
- Only **one** default constructor is allowed for each class

1.5.2 Default Constructors

- What is a default constructor?
  - a constructor that has no parameters
  - a constructor where all parameters have default arguments

- Characteristics of a default constructor:
  - it’s a member function of a class
  - only **one** default constructor can exist for each class
  - an empty default constructor is provided automatically
    - if no constructors are defined
  - it may be called explicitly

Default Constructors (cont.)

- Uses of default constructors:
  - to initialize the data members of an object with default values
- Default constructors are called *implicitly*:
  - when an object is declared
  - when memory for an object is dynamically allocated
    - using the **`new`** operator
    - ... more on this later...

1.5.3 Destructors

- Characteristics of a destructor:
  - it’s a member function of a class
  - only **one** destructor can exist for each class
  - it takes no arguments
  - an empty default destructor is provided automatically
  - it is **never** called explicitly (never on purpose by the programmer)
  - it is **always** called implicitly (automatically)

Destructors (cont.)

- Uses of destructors:
  - to perform clean-up when an object is destroyed
  - to release resources, if required
    - close any open files
    - release dynamically allocated memory
      - this will be our main usage of destructors in this course
      - ... more on this later...
    - whatever else is needed, it depends on the class
- You must write the destructor code yourself!
  - the compiler doesn’t know what clean-up is required

Destructors (cont.)

- Destructors are always called *implicitly*:
  - when a locally declared object moves out of scope
  - at the end of the program, for global objects
  - when dynamically allocated memory for an object is released
    - using the **`delete`** operator
    - ... more on this later...
  - they are usually called in reverse order of constructor calls

1.5.4 Order of Invocation

- For global objects:
  - constructor is called:
    - before the **`main()`** function begins
    - in order of declaration
  - destructor is called:
    - after the **`main()`** function terminates
    - if **`exit()`** is called
    - in reverse order of construction

Order of Invocation (cont.)

- For objects declared locally:
  - constructor is called:
    - every time the declaration is encountered
    - in a function, loop, any block
  - destructor is called:
    - when the object exits the scope
    - at the end of the block where the object is declared

Order of Invocation (cont.)

- Abnormal program termination
  - if **`exit()`** is called
    - causes immediate termination of program
    - destructors for global objects execute
  - if **`abort()`** is called
    - non-recoverable failure
    - causes immediate termination of program
    - no destructors execute

1.5.5 Copy Constructors

- Constructors that take one argument are special
  - copy constructors
    - constructor that takes a reference to an object of the same class
      - not an object of the same class
      - not a pointer to an object of the same class
    - can be called explicitly
    - also called implicitly in unexpected ways
  - conversion constructors
    - constructor that takes any type other than a reference to an object of the same class
    - ... more on this later ...

Copy Constructors (cont.)

- Characteristics of a copy constructor:
  - it's a member function of a class
  - it takes one parameter: reference to an object of the same class
  - copy constructor is provided automatically if none is specified
    - default copy constructor performs *member-wise assignment*
    - this is bad news for data members that are pointers
  - it may be called explicitly

Copy Constructors (cont.)

- Uses of copy constructors:
  - to make a copy of an existing object
  - to initialize a new object using values from an existing one

Copy Constructors (cont.)

- Some terminology:
  - declaration: `Student matilda;`
  - initialization: `Student matilda = bertha;`
  - assignment: `matilda = bertha;`

Copy Constructors (cont.)

- Copy constructors can be called explicitly:
  - on declaration when given a parameter value
- Copy constructors can be called implicitly:
  - when an object is passed by value as a function parameter
  - on **initialization**
    - do not confuse this with **assignment**!

1.5.6 Conversion Constructors

- Characteristics of a conversion constructor:
  - it's a member function of a class
    - for example, class A
  - it takes one parameter of a different data type
    - a data type not of the same class (not class A)
    - for example, class or data type B
  - conversion constructor is called when the program:
    - needs an object of the class (for example, A)
    - is only given a variable of the parameter data type (for example, B)
  - result: parameter is converted to new object using constructor

Conversion Constructors (cont.)

- Uses of conversion constructors:
  - they initialize the data members of a new object, using an object of another class
  - this is not always a good idea!

- The use of conversion constructors can be disabled:
  - use the **`explicit`** keyword
  - implicit uses will be disallowed
  - explicit uses will still be allowed