C++: Constructors and Destructors

Announcement

- RIT Career Fair
  - Thursday, Oct 3rd 1pm – 7pm
  - Friday, Oct 4th 9am – 5pm (interviews)
- Clark Gym
- www.rit.edu/co-op/careers

Announcement

- Date for Exam 1 will probably be changed
  - New date TBD.
  - Will know new date by Thursday.

Project

- Questions?
- Yes, there will be only one project.
  - Typo on Web page.
- Everyone have a partner?
- Please e-mail me with the name of your partner and I will assign you a group account.
- RCS in group accounts.

Plan for today

- Constructors
- Destructors
- Enumerated Types
- Assertions

Constructor

- A constructor for a class is called when an object of that class is created:
  - Local / Stack Based
    - `Foo F (3, 4, "Joe");`
  - On Free Store
    - `Foo *Fptr (new Foo ((3, 4, "Joe"));`

Constructor

- Constructors have the same name as the class.
- Constructors do not return a value.
- The constructor, like all functions, can be overloaded with each constructor having a different set of parameters.

Constructor

```
class Date {
private:
  int d, m, y;
public:
  Date (int day, int month, int year);
  Date (int day, int month);
  Date (int day);
  Date ();  // today's date
  Date (const *char stringdate);
  …
}
```

Constructor

```
Date::Date (int day, int month, int year)
{
  d = day;
  m = month;
  y = year;
}
```

Can also be written using *subobject constructor* (this way is more efficient).

```
Date::Date (int day, int month, int year) :
  d (day), m (month), y (year) {}
```

Constructor

- Default Constructor
  - Constructor with no arguments.
    - `Foo F;`
  - If a class defines constructors but no default constructor, compiler will generate an error.
  - If no constructors are defined for a class, the compiler will generate a default constructor.

Constructor

- Copy Constructor
  - Initializes an object based on the contents of another object of the same type.

```
Date (const Date &D) :
  d (D.d), m (D.m), y (D.y) {}
```

  - Object has access to non-public members of objects of the same class

Constructor

- Copy Constructor
  - Called when:
    - A declaration is made with *initialization from another object*
      - `Date d1 (d2);`
    - Parameters are passed by value.
    - An object is returned by a function.

Constructor

- Copy vs. Assignment
  - operator= is called when an assignment is made…
    - `Date d1 (10, 2, 2002);`
    - `Date d2;`
    - `d2 = d1;  // operator= called`
  - However,
    - If an assignment is made during a variable declaration, then the copy constructor rather than the assignment operator is called
      - `Date d1 = d2;  // Copy NOT assignment!`

Constructor

- Copy vs. Assignment

```
Date::Date (int day, int month, int year)
{
   d = day;    // constructor + assignment performed
   m = month;
   y = year;
}
```

Can also be written using *subobject constructor* (this way is more efficient).

```
Date::Date (int day, int month, int year) :
   d (day), m (month), y (year) {}
// just copy constructor is called.
```

Constructor

- Copy Constructor
  - If no copy constructor is defined for a class, the default copy constructor is used.
    - Member by member copy of data from one object to another.
    - Can be troublesome if class have pointers as data members.
  - Same issues as with the default assignment operator!!!!

Constructor Summary

```
Date d1(3, 10, 2002);  // constructor called
Date d2, d5;   // default constructor called
Date d3 (d2);  // copy constructor called
Date d4 = d1;  // copy constructor called
d5 = d2;       // assignment operator called.
```

Questions?

Constructor

- Important safety tips:
  - Always provide a default constructor
  - If your constructors perform any non-trivial work (e.g. memory allocation), should define the full suite of:
    - Constructors
    - Copy constructor
    - operator=

Destructor

- A destructor for a class is called when the memory of an object of that class is reclaimed:
  - A global (static) object is reclaimed when the program terminates.
  - A local (automatic) object is reclaimed when the function terminates (stack is popped).
  - A dynamically allocated object is reclaimed when someone invokes the `delete` operator on it.
  - Like Java `finalize`

Destructor

```
void aFunction (Foo f)
{
   Foo f2;
   Foo *fooptr = new Foo();
   ...
   delete fooptr; // destructor called
}
// after function is complete, destructor
  called on f and f2
```

Destructor

- Destructors have the same name as the class but preceded with a ~.
- Destructors do not return a value.
- Destructors take no arguments and cannot be overloaded.
- Destructors are used for cleaning up object data / state
  - Allocated memory
  - Close files
  - Close network connections, etc.

Destructors

```
class Foo
{
private:
    int *array_member;
    int asize;
    ...
public:
    Foo (int size);
    ~Foo ();
}
```

Destructors

```
Foo::Foo (int size) :
 asize (size), array_member (new int[size])
{
  for (int i=0; i<size; i++)
      array_member[i] = 0;
}

Foo::~Foo ()
{
  // cleanup what was allocated by
  // constructor
  if (array_member != 0) delete array_member;
}
```

Destructors

- Questions?

Enumerated Types

- An enumeration is a type that can hold a set of values defined by the user.
- Once defined, they can be used like an integer type.
- `enum` statement assigns sequential integer values to names and provide a type name for declaration.

Enumerated Types

```
enum TrafficLightColor {RED,
  YELLOW,GREEN};

TrafficLightColor x;
x = YELLOW;
```

Enumerated Types

- It's possible to control the values that are assigned to each enum constant.
  - `enum Day {MON=1, TUE, WED, THU, FRI, SAT, SUN};`
  - `WED == 3`
  - `enum DayFlag {MON=1, TUE=2, WED=4, THU=8, FRI=16, SAT=32, SUN=64};`

Enumerated Types

- Enums are types, not just integers

```
enum TrafficLightColor {RED, YELLOW, GREEN};
enum Gender {MALE, FEMALE};
TrafficLightColor x;
                . . .
x = YELLOW; // good
x = FEMALE; // error
x = 2;      // error
```

Enumerated Types

- Java doesn't have enums, instead, Java uses constants:

```
public int final MON=1;
public int final TUE=2;
public int final WED=3;
public int final THU=4;
public int final FRI=5;
public int final SAT=6;
public int final SUN=7;
```

Enumerated Types

- Enums can be placed in the scope of a class:

```
class Calendar
{
public:
 enum Day {MON=1, TUE, WED, THU, FRI, SAT, SUN};
...
}

Calendar::Day x;
x = Calendar::SAT;
```

Enumerated Types

- Questions?

Assertions

- Debugging mechanism to check condition at a given point in the code
  - If condition is false, then program will abort with an error message then dump core.
  - Detects code error, not user error.

Assertions

```
#include <cassert>

void f (int *p)
{
  // At this point p should be non-null
  assert (p!=0);
   …
}
```

Assertions

- Used for debugging
  - Can be turned off
  - Removing does not affect how the program works
  - `CC –DNDEBUG foo.C`

Assertions

- When to use
  - Test preconditions
  - Test postconditions
  - *"At this point x should be equal to …"*
  - *"We should never reach this line"*

Summary

- Constructors
  - Copy Constructor
  - Default Constructor
- Destructor
- Enum
- Assertion

- Questions?