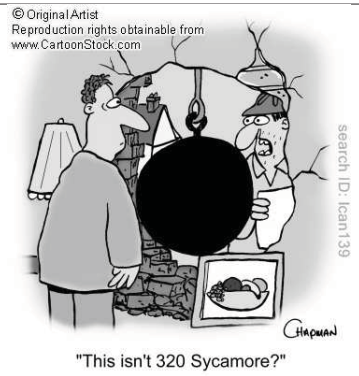


"This isn't 320 Sycamore?"

Slides #11 - Text 7.6-7.7

Constructors & Destructors
How to build objects (right)

Topics

1) How can we initialize an object when it's created?
2) How can we do things when an object is destroyed?
3) How can we write functions with different parameters?

Constructors

Constructor

- A constructor is:

- It's like other functions except:
  - its name must be
  - it has... (not even void!).

```
class Demo {
public:
    Demo() {
        cout << "Running Demo's constructor."<<endl;
    }
};
```

Constructor demo

```
class Demo {
public:
    Demo() {
        cout << "Running Demo's constructor."<<endl;
    }
};
int main(){
    cout << "Before instantiating."<<endl;
    Demo demoObj;
    cout << "After instantiating."<<endl;

    return 0;
}
```


Initialization

- Constructors are used to...

```
class BankBalance {
private:
    double balance;
public:
    BankBalance()
        { balance = 0; }

    double getBalance()
        { return balance; }
};
int main(){
    BankBalance myAcct;
    cout << fixed << setprecision(2);
    cout << "My account balance: " << myAcct.getBalance() << endl;
    ...
```


Inline vs not-inline

- Constructor can be inline, or non-inline.

Inline constructor:

```
class BankBalance {
private:
    double balance;
public:
    BankBalance()
        { balance = 0; }
};
```

Non-Inline constructor:

```
class BankBalance {
private:
    double balance;
public:
    BankBalance();
};


{
    balance = 0;
}
```


Review

- Implement Circle's constructor as a non-inline function. Initialize its size to 0.

```
class Circle {
private:
    double radius;
public:
    Circle();
    void setRadius(double r);
    double getArea();
};
```

Destructors

Destructor

- A destructor is:

- It's like other functions except:
  - its name is the...
  - it has no return type (not even void!)
  -

```
class Demo {
public:
    ~Demo() {
        cout << "Running Demo's destructor."<<endl;
    }
};
```


Destructor Demo

```
class Demo {
public:
    Demo() {
        cout << "+++ Running Demo's constructor."<<endl;
    }
    ~Demo() {
        cout << "--- Running Demo's destructor."<<endl;
    }
};
void testDemo() {
    cout << "1.  Before making the object."<<endl;
    Demo d1;
    cout << "2.  After making the object."<<endl;
}

int main(){
    testDemo();
    return 0;
}
```

```
1.  Before making the object.
+++ Running Demo's constructor.
2.  After making the object.
--- Running Demo's destructor.
```


Destructor Use

- Constructors run when created:
  - used for...
- Destructors run when being destroyed:
  - used for...
  - Examples:
    - closing a file.
    - freeing dynamically allocated memory.

Review

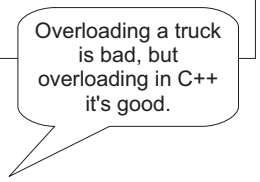
- Implement Circle's destructor as a non-inline function. Have it output the message "Bye".

```
class Circle {
private:
     double radius;
public:
     ~Circle();
     void setRadius(double r);
     double getArea();
};
```

Note: We will most often create constructors, but only rarely create destructors.

Overloading
Text 6.14

Function Overloading



- Function overloading:

- C++ can find the right function by its signature:
  -
  - Parameters:
    -
- Function Overloading Examples:
  - int sum (int a, int b) { return a + b; }
  - int sum (int a, int b, int c) { return a + b + c; }
  - int sum (bool x, int a) { if (x) { a; } else { 0; } }

Overloading examples

- Given:
  int sum (int a, char b);
-
  int sum (int a, long b); // Different type(s)
  int sum (char a, int b); // Diff. order of types
  int sum (int a, char b, int c); // Diff. # parameters
  int sum (); // Diff. # parameters
-
  long sum (int a, char b); //
  void sum (int a, char b); //
  int sum (int b, char a); //

Overloading Constructors

- Overloading is very useful for constructors:
  - You can create an object by passing in different sets of values.

```
class Drink {
public:
    Drink();
    Drink(string name);
    Drink(string name, bool alchoholic);
};
```


Default Constructor

- Default constructor:
  -
  - Automatically created for us if we specify no constructors for an object.

Demonstrate creating an object for each constructor.

```
class Drink {
public:
    Drink() { ... };
    Drink(string name) { ... };
    Drink(string name, bool alchoholic) { ... };
};
```

```
int main() {
    Drink d1;
    Drink d2("water");
    Drink d3("coke", false);
    return 0;
}
```


Demo / Review

- Implement a Pen class that stores the colour of ink. Make a default constructor, and a constructor to initialize the Pen's colour (a string).

Summary

- Constructors are functions to initialize objects.
  - Same name as class.
  - No return type.
- Destructors called when object being destroyed.
  - Similar to constructors but prefix name with ~ and cannot accept arguments.
- Overloading possible based on name, and parameters: #, order, and type.
  - Good for constructors.
- Default constructor is one with no parameters.