Class Constructors

[Sections 3.3-3.4]

Uninitialized Variables

- Default values for uninitialized variables are completely unpredictable

```
#include <iostream.h>

int main()
{
    int x1, x2;
    cout << x1 << " " << x2 << endl;
    return 0;
}
```

→ `0 1`

- Major source of bugs ("If you're lucky, the program will crash")
- This is fixed in Java

Uninitialized ADTs

```
…
Student someStudent;
someStudent.init( "Knox Overstreet" );
…
```

- What is value of **`someStudent`** before **`init`**?
- What if client doesn't call **`init`**?

```
…
NuclearReactor reactor;
reactor.init( SOME_SAFE_MODE );
…
```

- How do we force an instance to have reasonable initial state?

Invariants and Initialization

- Want to maintain invariant that instance data will always be "reasonable"
  - Separate declaration and initialization breaks this invariant
- Need mechanism for combining declaration and initialization

Constructors

- A constructor is a special C++ member function
  - Automatically called whenever a new instance is created
  - Name of member function is name of class
  - No return type, *not even void!*
- Cannot create an instance without invoking a constructor

Improved Student Class

```
…
class Student
{
public:
    Student();
    …
};
```

student.h

- New function added: **`Student::Student()`**
- No more **`init`** member function, everything else the same

Programming With Constructors

```
…
Student::Student()
{
    strcpy( name, "Debbie Default" );
    gpa = 2.0;
}
…
```

student.cpp

```
…
Student someStudent;
…
```

client.cpp

Implicitly calls
Student::Student()

Constructors With Arguments

- There isn't always a default value for an instance

```
class MailingAddress
{
public:
    MailingAddress();
    …
private:
    int number;
    char street[ 40 ];
    char city[ 40 ];
    State state;
    int zip;
};
```

- What's the "default mailing address"?
- Need a way to pass arguments to constructors

More Improved Student Class

```
…
class Student
{
public:
    Student( char str[] );
    …
};
```

student.h

```
class MailingAddress
{
public:
    MailingAddress( int nine_digit_zip[ 9 ] );
    …
};
```


More Programming With Constructors

```
…
Student::Student( char str[] )
{
    strcpy( name, str  );
    gpa = 4.0;
}
…
```

student.cpp

```
…
Student someStudent( "Irwin Initialized" );
…
```

client.cpp

Implicitly calls
Student::Student(char str[])

Recall…

```
#include <fstream.h>

…
ofstream outfile( "output.txt" );    // open output
…
```

- A peek inside **`ofstream`**…

```
class ofstream
{
public:
    …
    // Constructor that opens an output file
    ofstream( char filename[] );
    …
};
```

ofstream.h

Initializing Primitive Types

- Constructor syntax also applies to primitive types

```
int x( 15 );
double y( 197.6 );
bool b( false );
```

⟺

```
int x = 15;
double y = 197.6;
bool b = false;
```

- Careful - doesn't apply to arrays

```
char str[]( "A String" ); // No!
```

Default Constructors

- Compiler creates zero-argument constructor if none explicitly provided
  - But *only* if none explicitly provided
  - Call zero-argument constructors on data members
- If variable declaration does not explicitly call a constructor, empty constructor is assumed
- Empty constructors on primitive types do nothing (i.e., leave variables uninitialized!)
- Sometimes refers to *explicit* zero-argument constructor...

Multiple Constructors

- May be several reasonable ways to initialize a class instance
- So, use multiple constructors!
- All have same name (name of class)
- Distinguished by number and types of arguments
- Example of “overloading”

Multiple Constructor Example

```
class Student
{
    Student( char str[] );
    Student( int student_id );
    …
};
```

student.h

```
Student::Student( char str[] )
{
    // as before
}

Student::Student( int student_id )
{
    // Look up name in student database
}
```

student.cpp

`StudentCouncil` Revisited

```
class StudentCouncil
{
public:
    StudentCouncil( int sz,
        char prezname[], char mop_name[] );
private:
    int size;
    Student president;
    Student minister_of_propaganda;
};
```

council.h

- How can we call the constructors for **`president`** and **`minister_of_propaganda`** in the constructor for **`StudentCouncil`**?

Memberwise Initialization

- Constructors allow optional *memberwise initialization*: calls to constructors of data members
- Called even before body of constructor executes!

```
StudentCouncil::StudentCouncil(
     int sz, char prez[], char mop[] ) :
    size( sz ),
    president( prezname ),
    minister_of_propaganda( mop_name )
{}
```

council.cpp

Summary

- Avoid uninitialized data when possible
  - Causes unpredictable bugs
  - Breaks class invariants
- Constructors provide automatic initialization of new instances
  - Zero, one or more arguments
  - Multiple constructors in one class
  - If none provided, default is automatically generated
    - Default constructors for primitive types do nothing
  - Memberwise initialization