C++ Classes (I)

User-Defined Data Types

- Some programs can be written using only the built-in C++ data types
- Most programs create new data types that more effectively model the problem being solved
- Structs, enums, typedefs can be used to define new data types
- Classes can be used to create new data types that are fully integrated into the C++ language

User-Defined Data Types

```
// BigInteger supports arbitrary-precision integers
class BigInteger {
   …
};

int main() {
   BigInteger x("123451234512345");
   BigInteger y(789L);
   BigInteger z = -((x + y)/2);
   if (z < x) {
      ++z;
   }
   cout << z << endl;
}
```

Data Type Operations

- Before learning how to create new types with classes, let's look at how the built-in types are used
- Declare with no initial value

```
float f;
```

- Initialize with value of the same type

```
float f = 123.45f;
```

- Initialize with value of a different type

```
float f = 789;
```

- Assign to value of the same type

```
f = 123.45f;
```

Data Type Operations

- Assign to value of a different type

```
f = 789;
```

- Convert to a different type

```
double d = f;
```

- Apply operators

```
(f * g)
```

- Print to output stream

```
cout << f ;
```

- Deinitialize
  - Goes out of scope
  - Deallocated with delete
- Defining a new class involves specifying how all of these operations work on objects of the new class

String Class Example

- Let's define a class named String that is similar to the standard C++ string class

```
class String {
   …
};

void main() {
   String byu("BYU");
   String gatech;
   gatech.Append("Georgia Tech");
   String msg = gatech.Concat(" threw an interception.");
   msg.Write(cout);
   msg = byu.Concat(" kicked a field goal.");
   msg.Write(cout);
   for (int x=0; x < msg.Length(); ++x) {
      cout << msg.GetChar(x);
   }
}
```

Instance Variables

```
class String {

private:
   char * str;

};
```

Constructors

```
class String {

public:
   String() {
      Init("");
   }
   String(const char * s) {
      Init(s);
   }
   String(const String & s) {
      Init(s.str);
   }

private:
   void Init(const char * s) {
      str = new char[strlen(s) + 1];
      strcpy(str, s);
   }

};
```

Destructor

```
class String {

public:
   ~String() {
      Free();
   }

private:
   void Free() {
      delete [] str;
      str = 0;
   }

};
```

Assignment Operators

```
class String {

public:
   String & operator =(const char * s) {
      Free();
      Init(s);
      return *this;
   }

   String & operator =(const String & s) {
      if (&s != this) {
         Free();
         Init(s.str);
      }
      return *this;
   }

};
```

Instance Operations

```
class String {

public:
   bool IsEmpty() const {
      return (Length() == 0);
   }

   const char * CString() const {
      return str;
   }

   int Length() const {
      return strlen(str);
   }

};
```

Instance Operations

```
class String {

public:
   char GetChar(int index) const {
      return str[index];
   }

   void SetChar(int index, char c) {
      str[index] = c;
   }

};
```

Instance Operations

```
class String {

public:
   void Append(const char * s) {
      char * newStr = new char[strlen(str) + strlen(s) + 1];
      strcpy(newStr, str);
      strcat(newStr, s);
      Free();
      str = newStr;
   }

   void Append(const String & s) {
      Append(s.str);
   }

};
```

Instance Operations

```
class String {

public:
   String Concat(const char * s) const {
      String result(str);
      result.Append(s);
      return result;
   }

   String Concat(const String & s) const {
      return Concat(s.str);
   }

};
```

Instance Operations

```
class String {

public:
   void Write(ostream & s) const {
      s << str;
   }

};
```

Tracking the Number of String Objects

```
String * UseStrings() {
   String a[5];
   String b("a string");
   cout << String::ObjectCount() << endl;
   String * c = new String("another string");
   cout << String::ObjectCount() << endl;
   return c;
}

void main() {
   cout << String::ObjectCount() << endl;
   String * p = UseStrings();
   cout << String::ObjectCount() << endl;
   delete p;
   cout << String::ObjectCount() << endl;
}
```

Class (static) Variables

```
class String {

private:
   static int count;

};


int String::count = 0;
```

Modified Constructors

```
class String {

public:
   String() {
      Init("");
      IncObjectCount();
   }
   String(const char * s) {
      Init(s);
      IncObjectCount();
   }
   String(const String & s) {
      Init(s.str);
      IncObjectCount();
   }

private:
   static void IncObjectCount() {
      ++count;
   }

};
```

Modified Destructor

```
class String {

public:
   ~String() {
      Free();
      DecObjectCount();
   }

private:
   static void DecObjectCount() {
      --count;
   }

};
```

Class (static) Operations

```
class String {

public:
   static int ObjectCount() {
      return count;
   }

};
```

Compiler-Generated Members

- If a class doesn't provide them, the compiler will automatically generate the following members if it needs them
- Default (no-arg) constructor
- Copy constructor (member-wise initialization)
- operator = (member-wise assignment)
- Destructor