More on Class Design

Today's Objectives

- **Static** variables and methods (section 10.5)
- **Case Study**: the **string** class (section 10.2)

UML Class Design

Student
- int ID - int GPA + string Name + bool Female + Faculty Advisor
+ void Study() + void DoHomework() + bool Register(string name) + bool ChangeAdvisor(string + name) void Speak()

Faculty
+ string Name + string Office + string Telephone
+ void Teach() + void Advise()

name

properties

behaviors

- → private

+ → public

Instead of global variables, you can create **static** member variables/methods

```
class Student
{
private:
   int ID;
   static int IDPool;
public:
   Student() {
      ID = IDPool++;
   }
   static void SetIDPool(int id) {
      IDPool = id;
   }
};
```

Static variables:

1) Shared by all objects of that class

2) Exist without needing to create an object of that class

Rule: Static methods **cannot** access member variables, except static variables!

You can call a access static variable/ function with **class name** and "::" operator

Class name

```
int Student::IDPool = 0;

Student::SetIDPool(1000);
Student s1, s2;

cout << s1.Name << " has ID " << s1.GetID();
cout << endl;
```

The string class is a great example of Object Oriented Design

STRING

String member variables are private.

SUBSET OF THE STRING CLASS METHODS…

Member Functions

- size or length - Returns the number of characters in string.
- at - Accesses specified character with bounds checking.
- empty - Tests if a string is empty
- clear - Clears the contents
- insert - Inserts characters or string n times
- erase - Erases characters
- find - Search within a string
- append - Appends characters to the end
- replace - Replace characters of a string with another string
- resize - Changes the number of stored characters
- push_back - appends a character
- swap - Swaps the contents with another string

Member Operators

- operator[] - Accesses specified character
- operator+= Append to a a string
- operator== Check equality of strings, also !=, <, >, <=, >=, etc.

The properties (internal representation) are mostly "hidden" from the user.

The string class has a nice "interface" that we can use to manipulate string objects

You can create string objects with different constructors

- You can create an empty string using string's **default (no-arg) constructor**:

```
string newString;
```

- You can also create a string object using a **constructor that takes an argument**:

```
string name1("Klingon");
```

string::append() methods

append(string): appends **string** argument to the string

append(string, subpos, sublen):
appends **string** at position **subpos** for length **sublen**

append(n, char): appends **n** copies of the character **char** to the string

Overloaded functions/methods

Other useful string class methods

at(index): retrieve a character at a specified index

erase(index, n): delete part of the string

insert(index, string): insert anywhere in a string

replace(index, n, string): replace part of string

clear(): clear the string

empty(): test if a string is empty.