CSE 230
Intermediate Programming in C and C++
Classes and Data Abstraction

Fall 2017
Stony Brook University
Instructor: Shebuti Rayana

http://www3.cs.stonybrook.edu/~cse230/

Ref. Book: C How to Program, 8th edition by Deitel and Deitel
Constant Objects

■ The principle of “least privilege” can be applied to objects that are not modifiable.
■ The keyword `const` may be used to indicate that an object will not be modified after it is initialized.
■ Example:

```cpp
const Time noon(12, 0, 0);
```
■ C++ disallow any member function calls for `const` objects unless the functions themselves are declared `const`. This includes the `get` functions as well.
■ A function is specified both in its prototype and in its definition by inserting `const` after the parameter list.
■ Example:

```cpp
int Time::getHour() const { return hour; }
```
Constant Objects (cont.)

- An interesting problem arises for constructors and destructors, each of which often needs to modify objects.

- A constructor must be allowed to modify an object so that the object can be initialized properly.
  - A constructor is a non-constant member function that can be used to initialize a constant object.

- A destructor must be able to perform its termination housekeeping chores before an object is destroyed.

- The const declaration is not allowed for constructors and destructors.
const Data Member

■ A member constructor is used to initialize a private const data member.

■ The format is as follows:

```cpp
className::constructorName (parameter list)
    : privateDataName( value )
{ other statements }
```

■ For example:

```cpp
Increment::Increment(int c, int i)
    : increment(i)
{ count = c; }
```

■ All data members (including non-const) can be initialized using member constructor. For multiple initializations, include them in a comma-separated list after the colon.

■ For example:

```cpp
Increment::Increment(int c, int i) : increment(i), count(c) { }
```
Composition of Objects

- A class can have objects of other classes as members.

- Whenever an object is created, its constructor is called, so we need to specify how arguments are passed to member-objects constructors.

- Member objects are constructed in the order in which they are declared (not in the order they are listed in the constructor’s initializer list).

- Objects are constructed from the inside out and destructed in the reverse order.
Friend Function

- A **friend** function of a class is defined outside that class’s scope, yet has the right to access **private** members of the class.
- A function or an entire class may be declared to be a **friend** of another class.
- Using **friend** functions can enhance performance and it is often appropriate when a member function can not be used for certain operations.
- To declare a **friend** function, precede the function prototype with the keyword **friend**.
- To declare classTwo as a **friend** of classOne, place a declaration of the following form in the definition of classOne:

```cpp
friend class classTwo;
```
- Friendship is granted (not taken) and is neither symmetric nor transitive.
Using this

- Every object has access to its own address through a pointer called `this`.

- An object’s `this` pointer is not part of the object (has no effect in the `sizeof`. Rather, `this` is passed into the object (by the compiler) as an implicit first argument on every non-`static` member function.

- The `this` pointer is implicitly used to reference both the data members and member functions of an object. It can also be used explicitly.

- The type of the `this` pointer depends on type of object.
Dynamic Memory Allocation

- In C:

```
TypeName *typeNamePtr;
typeNamePtr = malloc(sizeof(TypeName));
```

- In C++ use `new` `typeName` to create a new space:

```
double *somePtr = new double(3.14);
int *arrayPtr = new int[10];
char *str = new char[20];
```

- Use `delete` `typeName` to destroy an allocated space:

```
delete somePtr;
delete [ ] arrayPtr;  // [ ] for arrays
```

- `new` and `delete` automatically call the class constructor and destructor respectively.
Static Class Members

- Each object of a class has its own copy of all the data members of the class.
- A **static** class variable is shared by all objects of a class and it represents “class-wide” information (i.e. a property of the class, not of a specific object).
- A **static** data member must be initialized once at file scope.
- Although **static** data members may seem like global variables, but they have class scope.
- A **static** member function has no **this** pointer and referring to it is a syntax error.
- The member function may be declared **static** if it does not access non-**static** class data members and member functions.