Object Oriented Programming CSI333

Object Oriented Programming Intro.: References:

1. CSI333 Lab Exercise 5; carefully read comments in lab code.

2. Stroustrup Date class example: Sections 10.1, 10.2.1-10.2.3, 10.2.7(intro), 10.2.8, 10.3

“Structs with member functions” can implement programmer defined data types. The full “Object Oriented Programming” idea adds inheritance, polymorphism and protection/information hiding.
A language (such as C++ or Java) that supports Object Oriented Programming allows programmers to create their own operations on types they create themselves, that work like built-in operations (such as +) on built-in types (such as ints).

The material on Object Oriented Programming in this course is summarized as follows:

1. `struct  classname {  ...  };` defines `classname` as the name of a programmer defined data type. The `struct` body can declare data members and prototypes of `member functions` aka `methods`.

2. A (non-static) member function is like an ordinary C++ function except it can be called only for a specific variable (or object) of the data type.

3. A C++ member function is defined with its class name in front of the function name, separated by the “class resolution operator” :: (double colon).

4. A (non-static) function member is called for a specific variable using the “dot” syntax for structure data member access:
   
   `variable . memfunction( arguments );`
   
   If `pin` is a pointer variable, we can code `pin->memfunction( arguments );`
5. During a call to a member function and within the function member body, names of data members refer to fields in the object for which the member function was called. Also, the keyword this denotes a pointer to that object.

Slide 4

When the member function is called, a pointer to the object is passed as a “hidden” parameter to make this work. Unlike with ordinary functions that programmers must remember to use with particular objects, the programmer does not have to explicitly code a pointer to the particular object as a parameter.

6. Static member functions of a class aka class methods (as opposed to instance methods) are invoked without any associated object.

Slide 5

7. A class as a MODULARIZATION UNIT:
Class (static) methods are operations natural WITHOUT reference to an instance.
Instance methods are operations that always apply to a particular object.
C++ classes can have friend functions. Java has no self-standing functions and no friends.
8. Except for constructors and destructors, member functions must be declared with a return type (could be void).

9. A constructor function is called when the associated object’s lifetime begins, be it automatic, static or free-store.

Constructors are named with the name of the struct or class that they belong to.

10. Java in a nutshell: Only primitive type variables and reference (POINTER) type variables have names. NO named class/struct variables! Suppose foo() is a member/method of class CLNAME.

C++: CLNAME *p = new CLNAME(); p->foo();

Java: CLNAME p = new CLNAME(); p.foo();

IN BOTH LANGUAGES, p is a POINTER VARIABLE!

In Java, CLNAME myCLNAME; does NOT construct a CLNAME instance, value(myCLNAME) is the NULL reference.
struct Node { Node *right; Node *left; Key val; };
Node *NodeP; Node *NodeQ; NodeP = new Node; ...;
NodeQ = NodeP;  //SHALLOW COPY

NodeP

NodeQ

Slide 8

Node *Node::deep(void) {
    Node *ret = new Node;
    if( this->right ) ret->right = right->deep();
    else ret->right = 0;
    if( left ) ret-> left = left->deep();
    else ret->left = 0;
    ret->val = val; return ret; }

NodeQ = NodeP->deep();  //MAKE A DEEP COPY

Slide 9
struct Node { Node *right; Node *left; Key val;
   Node *deep(void); }
Node *NodeP = new Node; // build data str.
Node *NodeQ = NodeP->deep();

Slide 10