Classes in C++

Follow principles established by OO paradigm

- A user-defined type
- A set of objects (instances) sharing the same storage structure and behavior
- A struct construct with operations
- Support for instance and class variables, instance and class methods

54

Class Definitions

- Class definition gives:
  1. name — unique identifier for the class
  2. instance variables — components of each instance (similar to struct fields)
  3. class variables — shared by all class instances
  4. instance methods — used by clients to work with instances
  5. class methods — used by clients to work with class

55

Syntax of Class Definitions

- Divided between two files:
  1. Header file (class-name.h) — declarations that are imported by clients of class-name
  2. Code file (class-name.cc) — additional definitions, must include class-name.h
- Syntax of class definition (in header file):
  Keyword class, followed by class name, followed by body containing variable and function declarations

```
class <class_name> {
...
} ;
```

56

Body of Class Header

- Define class so-called class members:
  - — data members, i.e., the variables
  - — member functions, i.e., the operations
- Data members defined by:
  - — member type
  - — member identifier

```
class DLList {
 DLNode head;
 int size;
 ... } ;
```

57

Body of Class Header

- Members functions:
  - — declared with function prototype (at least) in class body
  - — actual function definition will appear within the class of definition
  - — for inline member functions, give also function definition in class header (.h file)

58

Body of Class Definition

- Examples of member function declarations in class header:

```
class DLList {
  ...
  bool isEmpty() {return (size==0);}  // a message expression

  bool find_element(int x);
  bool insert_element(int x);
  DLList& sort();
 ... } ;
```

59

Class Scope

- Visibility rules intended to allow directed access to identifiers of data members and member functions within header file and code file
- General rules:
  - Member identifiers are visible through class definitions, including header and code files
  - Member identifiers follow scope rules based on nesting of units within units
- Consequence: Class-level definitions will:
  - hide outer-scope definitions of same identifier,
  - be hidden by inner-scope definitions of identifier

60

Class Scope

- Contrast visibility of members through class code with non-member identifiers, visible only from point where they are declared or defined

  Example: loop index variable defined in the loop header
- Class identifier declared by literal following class keyword, defined after end of class body
  - Possible to define pointer and reference identifiers of type C1 while defining C1, but not value identifiers
- Non-inline member functions defined at file scope in code file; however, assume they are nested within class

61

Definition of Member Functions

- Use scope operator in code file
- Scope operator has two formats:
  1. <class_name>::<member_id>
  2. ::<member_id>
- Use (1) to denote class member outside header file that defines class
- Use (2) to denote global identifier hidden by a local definition

62

Scope Operator ::

- Examples of operator use when defining non-inline member functions

```
int y;                          // file scope definition in class
                                // header file
class C1 {
int x, y;
void foobar(char*);
...
} ;
                                // in the code file
void C1::foobar(char* y) {
int z;
z = y ;             // parameter y, not file scope y
C1::y = 150 ;               // data member y, not file scope y
z = z + 10 * ::y;   // use file scope definition of y
...  }
```

63

Access to Class Members

- Each member has an access level that determines who has the right to access that member
- Three access levels are available:
  - private (default)

    This member is accessible only within the defining class
  - public

    This member is accessible everywhere (using a qualified name like a struct field)

64

Access to Class Members

- Third access level:
  - protected

    This member is accessible only within the defining class and its subclasses
- Private access is most restrictive, public is least
- Smalltalk: All variable identifiers are protected; all method identifiers are public
- Java: A variation of C++'s approach with 4 access levels

65

Access to Class Members

- General guidelines to conform with O-O paradigm and information hiding
  - Data members should generally be protected, sometimes even private
  - Member functions in class's protocol should generally be public
  - Auxiliary functions should be protected or private

66

Access to Class Members

- Impact on subclass code class libraries

  When writing subclasses of class C1, members defined protected and public by C1 are accessible in subclass's code but private members are not
- Syntax: keyword before each portion of class definition

```
class C1 {
public:                          // beginning of public portion
  double foo (double, double);
  void bar (int, int);
protected:                       // protected portion
   …
} ;
```

67