CSC 222: Computer Programming II

Spring 2004

- classes and objects
  - abstract data types, classes and objects
  - using existing classes, #include , member function calls
- user-defined classes
  - data fields, member functions
  - private vs. public, constructor, separate compilation
  - examples: Die, MarbleJar

Abstract data types (ADTs)

for primitive types such as numbers and characters,
libraries of stand-alone functions suffice (e.g., `cmath`, `cctype`)

for defining/manipulating more complex types,
a better (more generalizable) approach is needed

an *abstract data type (ADT)* is a collection of data and the associated operations that can be applied to that data

e.g., an integer is a number (series of digits) with assumed operations: addition, subtraction, multiplication, division, compare with other integers, …

e.g., a string is a sequence of characters enclosed in quotes with operations: concatenation, determine it length, access a character, access a substring, …

C++ classes and ADT's

in C++, new abstract data types can be defined as *classes*

- once a class has been defined and #included into a program, it is indistinguishable from the predefined data types of C++
  - ➔ by defining a new class, you can extend the expressibility of C++
- when you create an instance of a class, that *object* encapsulates the data and operations of that class (a.k.a. *member functions*)

EXAMPLE: the C++ string class in <string>

DATA: a sequence of characters (enclosed in quotes)

MEMBER FUNCTIONS:

```
+  >>  <<                        // operators for concatenation, input, and output
int length();                    // returns number of chars in string
char at(int index);              // returns character at index (first index is 0)
string substr(int pos, int len);// returns substring starting at pos of length len
int find(string substr);         // returns position of first occurrence of substr,
                                 //     returns constant string::npos if not found
int find(char ch);               // similarly finds index of character ch
. . .
```


Using a class

to use a class, must:

- #include its definition
- know how to create an object (constructor arguments?)
- know how to call its member functions

in general, to call a member function:

```
object.funtionCall
```

e.g.,

```
word.length()
word.at(0)
word.find(ch)
```

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

string Capitalize(string str);

int main()
{
    string word = "foobar";

    cout << word << " contains " << word.length()
         << " characters." << endl;

    cout << "Capitalized, it's: " << Capitalize(word) << endl;

    char ch;
    cout << "Enter a character to search for: ";
    cin >> ch;

    int index = word.find(ch);
    if (index == string::npos) {
        cout << word << " does not contain " << ch << endl;
    }
    else {
        cout << ch << " is found at index " << index << endl;
    }

    return 0;
}
//////////////////////////////////////////////////////////////
string Capitalize(string str)
// Assumes: str is a single word (no spaces)
// Returns: a copy of str with the first char capitalized
{
    char cap = toupper(str.at(0));
    return cap + str.substr(1, str.length()-1);
}
```

User-defined classes

similar to string, you can define your own classes in C++

example: suppose we wanted to conduct dice simulations

ugly code
- lots of detail
- repetition

also, we would like to make it easy to
- create dice with different # of sides
- keep track of # of rolls

```
// doubles.cpp
////////////////////////////////////////////

#include <iostream>
#include <ctime>
using namespace std;

int DieRoll(int numSides = 6);

int main()
{
    srand(unsigned(time(0)));

    int roll1 = -1;
    int roll2 = -2;

    int rollCount = 1;
    while (roll1 != roll2) {
        roll1 = DieRoll();
        roll2 = DieRoll();

        cout << "You rolled " << roll1
             << " and " << roll2 << endl;
        rollCount++;
    }

    cout << "It took " << rollCount << " roll(s)."
         << endl;

    return 0;
}
////////////////////////////////////////////////
int DieRoll(int numSides)
{
    return rand()%numSides + 1;
}
```


Die class

can encapsulate behavior of a die in a new class (type)

DATA: number of sides, count of number of rolls

OPERATIONS: roll the die, determine # sides, determine # rolls so far

in order to use a class (type), you must

- #include its definition

```
#include "Die.h"
```

quotes around file name specify a user-defined class (compiler will look in local directory)

- know how to create an object and apply the member functions

```
int Roll();
int NumSides();
int numRolls();
```

returns random roll of die
returns number of sides on die
returns number of times has been rolled

Dice simulation (with Die objects)

cleaner & more general

- hides ugly details of getting time, calling srand, and squeezing the random number into the range 1 to 6
- hides inclusion of low-level libraries
- makes it easy to create different sized dice, e.g.,

```
Die d1(8), d2(8);
```

- encapsulates the code for keeping track of # sides and # of rolls

```
// doubles.cpp
////////////////////////////////////////////

#include <iostream>
#include "Die.h"
using namespace std;

int main()
{
    Die d1, d2;

    int roll1 = -1;
    int roll2 = -2;

    while (roll1 != roll2) {
        roll1 = d1.Roll();
        roll2 = d2.Roll();

        cout << "You rolled " << roll1
             << " and " << roll2 << endl;
    }

    cout << "It took " << d1.NumRolls() << " roll(s)."
         << endl;

    return 0;
}
```


Die class implementation (overview)

In C++, a class definition defines the structure and behavior of a new type

```
class CLASSNAME
{
  public:
    MEMBER FUNCTION PROTOTYPES;
  private:
    DATA FIELDS;
};
```

data fields (or *instance variables*) maintain the state of the object

member functions (or *methods*) define the operations on the data

generally, member functions are *public* (meaning accessible to client programs), while data fields are *private* (meaning hidden from client programs)

```
class Die
{
  public:
    Die(int sides = 6);
    int Roll();
    int NumSides() const;
    int NumRolls() const;

  private:
    int rollCount;
    int numSides;
};
```

each class has a special member function called a *constructor*

- same name as class, no return type
- performs any necessary initializations for an object

Die.h

generally, a class definition is broken into 2 files

- header (.h) file contains the class definition w/ function prototypes
- implementation (.cpp) file contains the member function definitions

#ifndef, #define, #endif are preprocessor directives – ensure that the file doesn't get included more than once in a project

```
//  Die.h
//
//  A class for simulating an N-sided die.

#ifndef _DIE_H_
#define _DIE_H_

#include <cstdlib>
using namespace std;

class Die
{
  public:
    Die(int sides = 6);
    int Roll();
    int NumSides() const;
    int NumRolls() const;

  private:
    int rollCount;      // # times die rolled
    int numSides;       // # sides on die
};

#endif
```


Die.cpp

.cpp file contains the definitions of the member functions

- class membership is specified using ::

when creating a .NET project, all .cpp files must be added

- .NET performs smart compilation – .cpp files are only recompiled if they changed since the last compilation

*actually, Die.cpp contains additional code to ensure srand only called once

```
#include <ctime>
#include "Die.h"
using namespace std;

Die::Die(int sides = 6)
// constructor: initializes data fields
{
    srand(unsigned(time(0)));
    rollCount = 0;
    numSides = sides;
}

Die::int Roll()
// Returns: a random 'die' roll (and incr. count)
{
    int dieRoll = rand()%numSides + 1;
    rollCount++;

    return dieRoll;
}

Die::int NumSides() const
// Returns: # of sides on the die
{
    return numSides;
}

Die::int NumRolls() const
// Returns: # of times die has rolled
{
    return rollCount;
}
```

In-class exercise

- create a Win-32 Console Application named rolls (on T: drive)
- create a new files named `Die.h` and `Die.cpp`,cut-and-paste code from
  `www.creighton.edu/~davereed/csc222/Code/Die.h`
- `www.creighton.edu/~davereed/csc222/Code/Die.cpp`
- create a new file named `rolls.cpp`
  - write code that performs 10,000 dice rolls, counts & displays number of 7's
- add `rolls.cpp` and `Die.cpp` to project

Private data

information hiding (via private data & public member functions) is not just a good thing, it's a necessary thing for implementing ADTs

- ADT operations specify all ways to manipulate data
- must protect form unauthorized access

e.g., shouldn't be able to change the # of sides of an existing die

another example: Marble puzzles

Suppose a jar contains 50 black marbles & 50 white marbles.
If you draw 2 marbles at random, how likely is it that they are the same color?

if we wished to simulate a marble jar, what operations should it support?

MarbleJar class implementation

class definition provides:

- constructor in which the initial number of black & white marbles is specified
- member functions for adding & removing a marble, checking to see if jar is empty
- private data fields to keep track of number of marbles of each color

```
class MarbleJar
{
  public:
    MarbleJar(int black, int white);
    string DrawMarble();
    void AddMarble(string marble);
    bool IsEmpty() const;
  private:
    int numBlack, numWhite;
};
```


Separate compilation

for user-defined classes, it's good practice to divide the definition into 2 files:

- header (.h) file: contains class definition with member function prototypes
- implementation (.cpp) file: contains member function definitions

advantages

- header file is simple, serves as outline for the class
- .NET performs smart compilation – will compile .cpp files only as needed

```
// MarbleJar.h

#ifndef _MARBLEJAR_H_
#define _MARBLEJAR_H_

#include <string>
using namespace std;

class MarbleJar
{
  public:
    MarbleJar(int black, int white);
    string DrawMarble();
    void AddMarble(string marble);
    bool IsEmpty() const;
  private:
    int numBlack, numWhite;
};

#endif
```

MarbleJar.cpp

.cpp file contains member function definitions

- class ownership specified using ::

note: DrawMarble uses a Die object

- think of all marbles as numbered, black first
- uses Die roll to pick a random marble number
- then determines if number corresponds to black or white

```
// Marblejar.cpp

#include "Die.h"
#include "MarbleJar.h"

MarbleJar::MarbleJar(int black, int white)
// precondition : black >= 0, white >= 0
// postcondition: numBlack = black, numWhite = white
{
    numBlack = black;
    numWhite = white;
}

string MarbleJar::DrawMarble()
// precondition : jar is not empty (numBlack+numWhite > 0)
// postcondition: draws a random marble and returns its color
//                (either "black" OR "white").  As a side effect,
//                the corresponding color count is decremented.
{
    Die myDie(numBlack+numWhite);
    if (myDie.Roll() <= numBlack) {
        numBlack--;
        return "black";
    }
    else {
        numWhite--;
        return "white";
    }
}
```


MarbleJar class implementation (cont.)

adding a marble simply increments the corresponding data field

what if an invalid marble color is specified?

```
// CONTINUING Marblejar.cpp

void MarbleJar::AddMarble(string marble)
// precondition : marble == "black" or "white"
// postcondition: adds a marble of the specified color to the jar,
//                incrementing the color count.
{
    if (marble == "black") {
        numBlack++;
    }
    else if (marble == "white") {
        numWhite++;
    }
}

bool MarbleJar::IsEmpty() const
// precondition : jar contains numBlack and numWhite marbles
// postcondition: returns true if jar is empty (no marbles)
{
    return (numBlack + numWhite == 0);
}
```

MarbleJar simulation

here, use the assert function from <cassert> to verify that the jar is not initially empty

```
// marbles.cpp

#include <iostream>
#include <string>
#include <cassert>
#include "MarbleJar.h"
using namespace std;

const int NUM_REPS = 100000;

int main()
{
    int numMarbles;

    cout << "Enter initial number of marbles of each color: ";
    cin >> numMarbles;

    assert(numMarbles >= 1);

    int sameCount = 0;
    for (int rep = 0; rep < NUM_REPS; rep++) {
          MarbleJar jar(numMarbles, numMarbles);

          if (jar.DrawMarble() == jar.DrawMarble()) {
                sameCount++;
          }
    }

    cout << "The marbles were the same " << 100.0*sameCount/NUM_REPS
         << "% of the time" << endl;

    return 0;
}
```


Enumerated types

could use enum to define a Color type

- safer, since limits legal values
- unfortunately, can't output directly

```
enum Color { BLACK, WHITE };

class MarbleJar
{
  public:
    MarbleJar(int black, int white);
    Color DrawMarble();
    void AddMarble(Color marble);
    bool IsEmpty() const;
  private:
    int numBlack, numWhite;
};

////////////////////////////////////////

#include "Die.h"
#include "MarbleJar.h"

MarbleJar::MarbleJar(int black, int white)
{
    numBlack = black;
    numWhite = white;
}
```

```
Color MarbleJar::DrawMarble()
{
    Die myDie(numBlack+numWhite);
    if (myDie.Roll() <= numBlack) {
        numBlack--;
        return BLACK;
    }
    else {
        numWhite--;
        return WHITE;
    }
}

void MarbleJar::AddMarble(Color marble)
{
    if (marble == BLACK) {
        numBlack++;
    }
    else {
        numWhite++;
    }
}

bool MarbleJar::IsEmpty() const
{
    return (numBlack + numWhite == 0);
}
```

In-class exercise

- create a Win-32 Console Application named marbles (on T: drive)
- create new files and cut-and-paste code from
  www.creighton.edu/~davereed/csc222/Code/Die.h
  www.creighton.edu/~davereed/csc222/Code/Die.cpp
  www.creighton.edu/~davereed/csc222/Code/MarbleJar.h
  www.creighton.edu/~davereed/csc222/Code/MarbleJar.cpp
  www.creighton.edu/~davereed/csc222/Code/marbles.cpp
- add MarbleJar.cpp, Die.cpp, and marbles.cpp to the project

QUESTION: does the number of marbles affect the percentage? How?