CS 103 Unit 14

Classes Revisited

Mark Redekopp

UML (Unified Modeling Language)

- Shows class definitions in a language-agnostic way
- Shows class hierarchy (inheritance, etc.)
- Each class shown in one box with 3 sections
  - Class Name, Member functions, then Data members
  - Precede function/data member with:
    + (public), - (private), # (protected)
  - Functions show *name with arguments* : *return type*
  - Data members show *name* : *type*

class name (e.g. Deck)
Member functions + shuffle() : void + cut() : void + get_top_card() : int
Member data - cards[52] : int - top_index : int

```
class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   void shuffle();
   void cut();
   int get_top_card();
 private:
   int cards[52];
   int top_index;
};
```

Class Notes

- Remember data members live on from one member function call to the next and can be accessed within ANY member function

```
#include <iostream>
#include <vector>
using namespace std;

class ABC
{
  public:
   ABC();
   void add_score(int s);
   int get_score(int loc); private:
   vector<int> scores;
};

// A change to scores here
void ABC::add_score(int s){
  scores.push_back(s);
}

// would be seen by subsequent
//  calls to member functions
int ABC::get_score(int loc){
  return scores[loc];
}

int main(){
  ABC a;
  a.add_score(95);
  a.get_score(0);
}
```

Class Design

- Class names should be 'nouns'
- To decide what objects/classes you need use
  - Object discovery: Based on the requirements of description of the problem look for the nouns/object
  - Object invention: Objects that simplify management or help glue together the primary objects
- Method/Function names should be 'verbs'

http://www.cprogramming.com/tutorial/class_design.html

```
class GradeBook
{
  public:
   computeAverage();
   int* getScores()
     { return scores; }
  private:
   int scores[20];
   int _size;
};

bool GradeBook::computeAverage(){
  double sum = 0.0;
  for(int i=0; i < _size; i++){
   sum += scores[i];
  }
  return sum / _size;
}


int main()
{
  GradeBook gb;
  int* myscores = gb.getScores();
  double sum = 0.0;
  for(int i=0; i < _size; i++){
   sum += myscores[i];
  }
  ...
}
```

Class Design

- Keep the computation where the data is (i.e. in the appropriate class member functions)

```
class GradeBook
{
  public:
   double computeAverage();
   int* getScores()
     { return scores; }
   int size() { return _size; }
  private:
   int scores[20];
   int _size;
};

double GradeBook::computeAverage(){
  double sum = 0.0;
  for(int i=0; i < _size; i++){
   sum += scores[i];
  }
  return sum / _size;
}

int main()
{
  GradeBook gb;
  int* myscores = gb.getScores();
  double sum = 0.0;
  for(int i=0; i < gb.size(); i++){
   sum += myscores[i];
  }
  ...
}
```

Headers & using statements (1)

deck.h

Won't compile.
Confused by 'vector'

```
#include <vector>
#include <string>

class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   void shuffle();
   void cut();
   string cardName(); // "2H, 9D"
 private:
   vector<int> cards;
   int top_index;
};
```

Option 1: Add 'using' statement BAD!!

```
#include <vector>
#include <string>
using namespace std;
class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   void shuffle();
   void cut();
   string cardName(); // "2H, 9D"
 private:
   vector<int> cards;
   int top_index;
};
```

Option 2: Qualify class names with 'std::' GOOD!

```
#include <vector>
#include <string>

class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   void shuffle();
   void cut();
   std::string cardName(); // "2H, 9D"
 private:
   std::vector<int> cards;
   int top_index;
};
```

Headers & using statements (2)

- By putting 'using' statements in a header file, anyone who #includes your header is now automatically using that namespace
- Rule: **NEVER** put a **'using'** statement in a *header (.h) file*

Option 1: Add 'using' statement BAD!!

```
#include <vector>
#include <string>
using namespace std;
class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   void shuffle();
   void cut();
   string cardName(); // "2H, 9D"
 private:
   vector<int> cards;
   int top_index;
};
```

deck.h

```
#include "deck.h"

class string {
  // My own implementation of 'string'
};

int main()
{
  string x; // which string is used?
            // The one from this file
            // or std::string
}
```

Headers & `using` statements (3)

- Corollary: FINE to put 'using' statements in .cpp files (since we don't #include .cpps)

Option 2: Qualify class names with 'std::' GOOD!

```
#include <vector>
#include <string>

class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   void shuffle();
   void cut();
   std::string cardName(); // "2H, 9D"
 private:
   std::vector<int> cards;
   int top_index;
};
```

deck.h

```
#include "deck.h"
using namespace std;

void Deck::shuffle()
{
 /* code */
}

string Deck::cardName()
{

}
```

deck.cpp

Final, Preferred Solution

- Header files
  - No 'using' statements in header files
  - Instead precede C++ class types with 'std::'
- CPP files
  - Add 'using' statements as desired
  - If you do add 'using' statement then no need to precede type with 'std::'

```
#include <vector>
#include <string>

class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   void shuffle();
   void cut();
   std::string cardName(); // "2H, 9D"
 private:
   std::vector<int> cards;
   int top_index;
};
```

Option 2: Qualify class names with 'std::' GOOD!

```
#include "deck.h"

class string {
  // My own implementation of 'string'
};

int main()
{
  string x; // No confusion. Use this file's
            // string
}
```