CS 103 Unit 10 Slides

C++ Classes

Mark Redekopp

Object-Oriented Approach

- Model the application/software as a set of objects that interact with each other
- Objects fuse data (i.e. variables) and functions (a.k.a methods) that operate on that data into one item (i.e. object)
- Objects replace global-level functions as the primary method of encapsulation and abstraction
  - Encapsulation: Hiding implementation and controlling access
    - Group data and code that operates on that data together into one unit
    - Only expose a well-defined interface to control misuse of the code by other programmers
  - Abstraction
    - Hiding of data and implementation details
    - How we decompose the problem and think about our design at a higher level rather than considering everything at the lower level

Object-Oriented Programming

- Objects contain:
  - Data members
    - Data needed to model the object and track its state/operation (just like structs)
  - Methods/Functions
    - Code that operates on the object, modifies it, etc.
- Example: Deck of cards
  - Data members:
    - Array of 52 entries (one for each card) indicating their ordering
    - Top index
  - Methods/Functions
    - `shuffle(), cut(), get_top_card()`

C++ Classes

- Classes are the programming construct used to define objects, their data members, and methods/functions
- Similar idea to structs
- Steps:
  - Define the class' data members and function/method prototypes
  - Write the methods
  - Instantiate/Declare object variables and use them by calling their methods
- Terminology:
  - Class = Definition/Blueprint of an object
  - Object = Instance of the class, actual allocation of memory, variable, etc.

cardgame.cpp

```
#include <iostream>
using namespaces std;
class Deck {
 public:
   Deck();   // Constructor
   int get_top_card();
 private:
   int cards[52];
   int top_index;
};

// member function implementation
Deck::Deck()
{
  for(int i=0; i < 52; i++)
    cards[i] = i;
}
int Deck::get_top_card()
{
   return cards[top_index++];
}
// Main application
int main(int argc, char *argv[]) {
  Deck d;
  int hand[5];
  d.shuffle();
  d.cut();
  for(int i=0; i < 5; i++){
    hand[i] = d.get_top_card();
  }
  ...
```

Common C++ Class Structure

- Common to separate class into separate source code files so it can easily be reused in different applications
- Steps:
  - Define the class':
    1.) data members and
    2.) function/method prototypes
    (usually in a separate header file)
  - Must define the class using the syntax:
    - `class name { ... };`
  - Write the methods **(usually in a separate .cpp file)**
  - **Instantiate/Declare object variables** and use them by calling their methods

deck.h

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

deck.cpp

```
#include<iostream>
#include "deck.h"


// Code for each prototyped method
```

cardgame.cpp

```
#include<iostream>
#include "deck.h"

int main(int argc, char *argv[]) {
  Deck d;
  int hand[5];

  d.shuffle();
  d.cut();
  for(int i=0; i < 5; i++){
    hand[i] = d.get_top_card();
  }
}
```

Access Specifiers

- Each function or data member can be classified as public, private, or protected
  - These classifications support encapsulation by allowing data/method members to be inaccessible to code that is not a part of the class (i.e. only accessible from within a public class method) to avoid mistakes by other programmers
  - Ensure that no other programmer writes code that uses or modifies your object in an unintended way
  - **Private**: Can call or access only by methods/functions that are part of that class
  - **Public**: Can call or access by any other code
  - **Protected**: More on this in CS 104
- Everything private by default so you must use `public:` to make things visible
- Make the interface public and the guts/inner-workings private

deck.h

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

deck.cpp

```
#include<iostream>
#include "deck.h"


// Code for each prototyped method
```

cardgame.cpp

```
#include<iostream>
#include "deck.h"

int main(int argc, char *argv[]) {
  Deck d;
  int hand[5];

  d.shuffle();
  d.cut();

  d.cards[0] = ACE; //won't compile
  d.top_index = 5;  //won't compile
}
```

Constructors / Destructors

- **Constructor** is a function of the same name as the class itself
  - It is called automatically when the object is created (either when declared or when allocated via 'new')
  - Use to initialize your object (data members) to desired initial state
  - **Returns nothing**
- **Destructor** is a function of the same name as class itself with a '~' in front
  - Called automatically when object goes out of scope (i.e. when it is deallocated by 'delete' or when scope completes)
  - Use to free/delete any memory allocated by the object or close any open file streams, etc.
  - **Returns nothing**
  - **[Note: Currently we do not have occasion to use destructors; we will see reasons later on in the course]**

deck.h

```
class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   ...
};
```

deck.cpp

```
#include<iostream>
#include "deck.h"

Deck::Deck() {
  top_index = 0;
  for(int i=0; i < 52; i++){
    cards[i] = i;
  }
}
Deck::~Deck() {
}
```

cardgame.cpp

```
#include<iostream>
#include "deck.h"

int main() {
  Deck d;  // Deck() is called
  ...

  return 0;
  // ~Deck() is called since
  // function is done
}
```

Writing Member Functions

- What's wrong with the code on the left vs. code on the right

```
void f1()
{
  top_index = 0;
}
```

```
Deck::Deck()
{
  top_index = 0;
}
```

- Compiler needs to know that a function is a member of a class
- Include the name of the class followed by '**::**' just before name of function
- This allows the compiler to check access to private/public variables
  - Without the scope operator [i.e. `int get_top_card()` rather than `int Deck::get_top_card()`] the compiler would think that the function is some outside function (not a member of Deck) and thus generate an error when it tried to access the data members (i.e. cards array and top_index).

deck.h

```
class Deck {
 public:
   Deck();   // Constructor
   ~Deck();  // Destructor
   void shuffle();
   ...
};
```

deck.cpp

```
#include<iostream>
#include "deck.h"
Deck::Deck() {
  top_index = 0;
  for(int i=0; i < 52; i++){
    cards[i] = i;
  }
}
Deck::~Deck()
{
}

void Deck::shuffle()
{
  cut(); //calls cut() for this object
  ...
}
int Deck::get_top_card()
{
  top_index++;
  return cards[top_index-1];
}
```

Multiple Constructors

- Can have multiple constructors with different argument lists

```
class Student {
 public:
   Student();   // Constructor 1
   Student(string name, int id, double gpa);
                // Constructor 2
   ~Student();  // Destructor
   string get_name();
   int get_id();
   double get_gpa();

   void set_name(string name);
   void set_id(int id);
   void set_gpa(double gpa);
 private:
   string name_;
   int id_;
   double gpa_;
};
```

student.h

Note: Often name data members with special decorator (id_ or m_id) to make it obvious to other programmers that this variable is a data member

```
#include<iostream>
#include "student.h"

int main()
{
  Student s1;  // calls Constructor 1
  string myname;
  cin >> myname;
  s1.set_name(myname);
  s1.set_id(214952);
  s1.set_gpa(3.67);

  Student s2(myname, 32421, 4.0);
             // calls Constructor 2

}
```

```
Student::Student()
{
  name_ = "", id_ = 0; gpa_ = 2.0;
}

Student::Student(string name, int id, double gpa)
{
  name_ = name; id = id_; gpa = gpa_;
}
```

student.cpp

Accessor / Mutator Methods

- Define public "get" (accessor) and "set" (mutator) functions to let other code access desired private data members
- Use 'const' after argument list for accessor methods
  - Ensures data members are not altered by this function (more in CS 104)

```
#include<iostream>
#include “deck.h”

int main()
{
  Student s1;  string myname;
  cin >> myname;
  s1.set_name(myname);

  string another_name;
  another_name = s1.get_name();

  ...
}
```

student.h

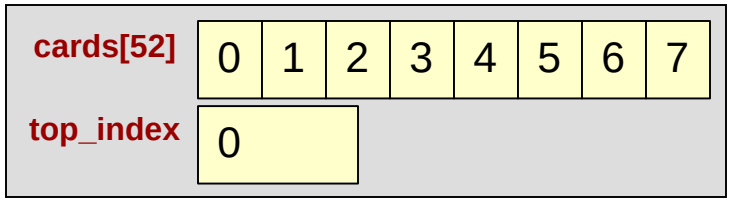
```
class Student {
 public:
   Student();   // Constructor 1
   Student(string name, int id, double gpa);
                // Constructor 2
   ~Student();  // Destructor
   string get_name() const;
   int get_id() const;
   double get_gpa() const;

   void set_name(string s);
   void set_id(int i);
   void set_gpa(double g);
 private:
   string _name;
   int _id;
   double _gpa;
};
```

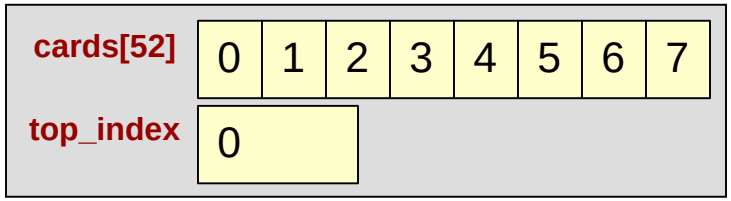
student.cpp

```
string Student::get_name() const
{   return _name; }
int Student::get_id() const
{   return _id; }
void Student::set_name(string s)
{  _name = s;  }

void Student::set_gpa(double g)
{  _gpa = g; }
```

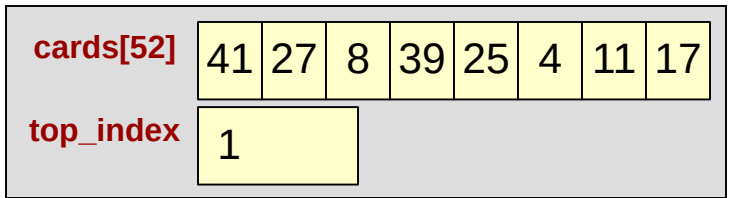
Calling Member Functions (1)

- When **outside the class scope** (i.e. in main or some outside function)
  - Must precede the member function call with the name of the specific object that it should operate on (i.e. `d1.memfunc()`)
  - `d1.shuffle()` indicates the code of `shuffle()` should be operating implicitly on d1's data member vs. d2 or any other Deck object






```
#include<iostream>
#include "deck.h"

int main() {
  Deck d1, d2;
  int hand[5];

  d1.shuffle();
  // not Deck.shuffle() or
  // shuffle(d1), etc.

  for(int i=0; i < 5; i++){
    hand[i] = d1.get_top_card();
  }
}
```



Calling Member Functions

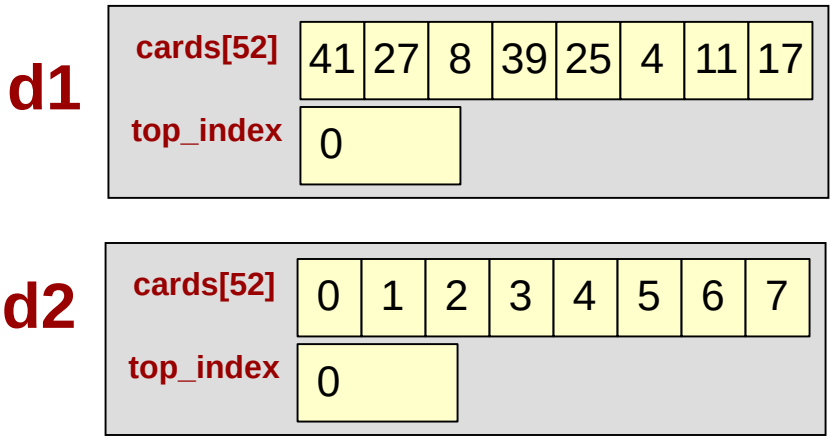
- When **inside the class scope** (i.e. in main or some outside function)
- Within a member function we can just call other member functions directly.

```
#include<iostream>
#include "deck.h"

int main(int argc, char *argv[]) {
  Deck d1, d2;
  int hand[5];

  d1.shuffle();
  ...
}
```

poker.cpp

d1's data will be modified (shuffled and cut)

d1 is implicitly passed to shuffle()




```
#include<iostream>
#include "deck.h"

void Deck::shuffle()
{
  cut(); // calls cut()
         // for this object
  for(i=0; i < 52; i++){
    int r = rand() % (52-i);
    int temp = cards[r];
    cards[r] = cards[i];
    cards[i] = temp;
  }
}
void Deck::cut()
{
  // swap 1st half of deck w/ 2nd
}
```

deck.cpp

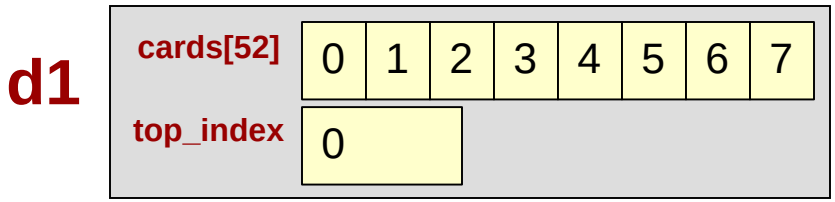
Since shuffle was implicitly working on d1's data, d1 is again implicitly passed to cut()

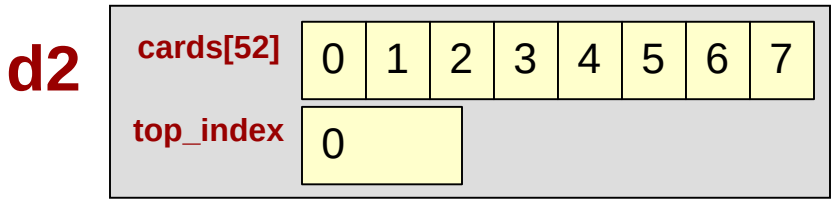
Exercises

- In-class Exercises

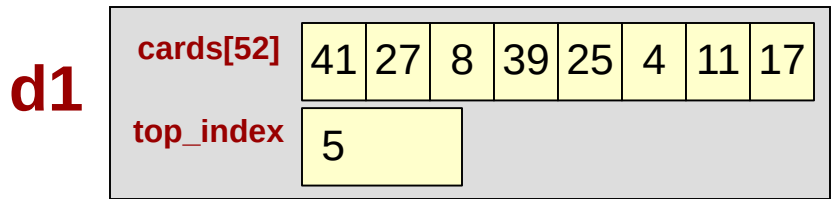
Class Pointers

- Can declare pointers to these new class types
- Use '->' operator to access member functions or data






```
#include<iostream>
#include "deck.h"

int main(int argc, char *argv[]) {
  Deck *d1;
  int hand[5];

  d1 = new Deck;

  d1->shuffle();
  for(int i=0; i < 5; i++){
    hand[i] = d1->get_top_card();
  }
}
```



Public / Private and Structs vs. Classes

- In C++ the only difference between structs and classes is structs default to public access, classes default to private access
- Thus, other code (non-member functions of the class) ***cannot*** access private class members directly

student.h

```
class Student {  // what's the difference
struct Student { // between these two

   Student();   // Constructor 1
   Student(string name, int id, double gpa);
                // Constructor 2
   ~Student();  // Destructor
   ...
   string name_;
   int id_;
   double gpa_;
};
```

grades.cpp

```
#include<iostream>
#include "student.h"
int main()
{
  Student s1;  string myname;
  cin >> myname;
  s1._name = myname;
    // compile error if 'class' but not
    // if 'struct'
  ...
}
```