Object Orientation

Why We Do It

Typical "Early" Code

- Often, when programming is taught, the majority of the focus is on learning to use basic data types, programming logic, and functions.
- Much of the program is often thrown into one main method, which might call one or two other functions.

Typical "Early" Code

- It's readily apparent that any program has (at least) two fundamental component categories that the user must define and manage.
  - –Data – the information received, output, and maintained by the program
  - –Functions/Methods – the programming logic that manipulates data as needed.

Typical "Early" Data

- For (nearly) any program to serve a useful purpose, it will need to meaningfully store and use some type of data.
  - What are some of the basic data types that you've used to this point in programming?

Typical "Early" Data

- int; long; char; float; double; bool; string; and array [];
  - Combination of these types can represent more complex types
  - Two basic types: integers and float
  - Integers are whole numbers
  - Integers can be signed or unsigned

C++ vs. C languages

```
#include <iostream>

using namespace std;

class cl { int i; // private by default
public: int get_i(); int put_i(int j); }

int cl::get_i() {return i;}
int cl::put_i(int j) {i = j;}

int main()
{
  cl s;
  s.put_i(10);
  cout << s.get_i() <<endl;
  return 0;
}
```

```
#include <iostream>

int main()
{
int I, j=10;

  i=j;
  printf ("%d \n");
  return 0;
  }
```

A Rough Exercise

- Suppose we wanted to write a program for playing a card game of some sort.
  - Like with Hearts or Spades, the full deck is dealt to four players.
- Disregarding the rules of the game… how would we *manage the cards*?

A Rough Exercise

- Cards are important information / data to keep track of for a card game.
- What manipulates cards, and how would this have to be coded?
  - Shuffling
  - Dealing
  - Each player has a separate hand…

A Rough Exercise

- The following program illustrates both two-dimensional arrays and constant arrays.
- The program deals a random hand from a standard deck of playing cards.
- Each card in a standard deck has a *suit* (clubs, diamonds, hearts, or spades) and a *rank* (two, three, four, five, six, seven, eight, nine, ten, jack, queen, king, or ace).

A Rough Exercise

- The user will specify how many cards should be in the hand:

```
   Enter number of cards in hand: 5
   Your hand: 7c 2s 5d as 2h
```

- Problems to be solved:

```
   * How do we pick cards randomly
from the   deck?
   * How do we avoid picking the same
card twice?
```

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUM_SUITS 4
#define NUM_RANKS 13

int main(void)
{
  bool in_hand[NUM_SUITS][NUM_RANKS] = {false};
  int num_cards, rank, suit;
  const char rank_code[] =
{'2','3','4','5','6','7','8',

'9','t','j','q','k','a'};
  const char suit_code[] = {'c','d','h','s'};
```

```
srand((unsigned) time(NULL));

  printf("Enter number of cards in hand: ");
  scanf("%d", &num_cards);

  printf("Your hand:");
  while (num_cards > 0) {
    suit = rand() % NUM_SUITS;    /* picks a random suit */
    rank = rand() % NUM_RANKS;    /* picks a random rank */
    if (!in_hand[suit][rank]) {
      in_hand[suit][rank] = true;
      num_cards--;
      printf(" %c%c", rank_code[rank], suit_code[suit]);
    }
  }
  printf("\n");

  return 0;
}
```

A Rough Exercise

- Now, consider the complexity of what we've put forth.
- There were many servers for competitive card-game playing
  - Imagine having to code like this for *thousands of simultaneous games*
  - How would *that* work?

Motivation

- One of the most evident problems that arises in novice programming is a lack of scalability.
  - This is often fine for initial learning – simplicity leaves much less room for confusion.
  - The more interesting question – why is the typical novice programming style not scalable?

Motivation

- Two key things to note in novice-style coding:
  - Note how we're organizing data.
  - Note how we're accessing data in the various functions of our proposed programs.

Motivation

- Two key things to note in novice-style coding:
  - Note how we're organizing data.
  - How is the data grouped together?
  - Do these groupings help clarify things?
  - Are we limited to a fixed size/count of data?
  - Note how we're accessing data…

Motivation

- Two key things to note in novice-style coding:
  - Note how we're organizing data.
  - Note how we're accessing data…
  - Do we have to copy-paste code to multiple points of our program, with slight modifications each time?
  - Do we have to assume all code copies operate perfectly for any of our code to work correctly?

Object Orientation

- The coding style of object-orientation provides one popular solution to these concerns.
  - Data are organized to represent distinct *objects* of the scenario being modeled.
    - The card deck
    - Each player's hand
    - Each individual card
  - This is done by defining *custom* data types.

Object Orientation

- The coding style of object-orientation provides one popular solution to these concerns.
  - When these conceptual "objects" of the program are modeled as custom data types, we may then manipulate them through functions designed to operate upon those custom types.
  - CardHand[] CardDeck::dealHands
  - (int numHands, int numCards)

Object Orientation

- The coding style of object-orientation provides one popular solution to these concerns.
  - Additionally, we may provide some functionality that will be seen as *inherent* to these custom data types.
  - These allow accessing and manipulating attributes of our program's objects.
  - `void CardDeck::shuffle();`

Object Orientation

- We don't think about it like this, but such functions already exist for our basic data types…
  - 1 + 1
  - 3.14159 * 2.71828
  - From Java:
  - "Hello " + "World"
  - System.*out*.println("The answer is " + 42);
  - As written, these do not translate directly into C++.
  - In C++, cout << "The answer is " << 42 << endl;

Object Orientation

- Programming then becomes about recognizing the distinct "objects" that need to exist within the system and coding them appropriately.
  - This includes needed interactions among objects.