Data Structures and OOP

Section 1D

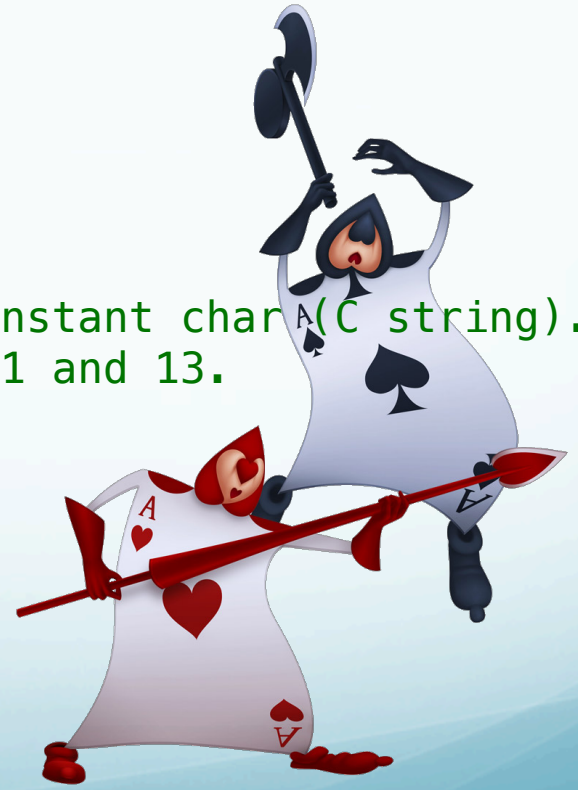
Structures

- A structure is a collection of related variables that may be of several different data types.

User-defined data type

```
struct Card
{
    const char *suit;   // A pointer to a constant char (C string).
    int face;           // A number between 1 and 13.
};
```

Properties or data members



Using `struct` data types

```
Card c1;                // Create a card
c1.suit = "Hearts";     // Fill out data members
c1.face = 3;
```

dot operator

```
Card c2;                // Create another card
c2 = c1;                // Assign member-to-member
c2.suit = "Spades";
```

assignment

```
Card deck[52];          // An array of cards
deck[0].suit = "Diamonds";
deck[0].face = 1;
```

arrays

Using `struct` data types

```
Card *cPtr;         // A pointer to a card
cPtr = &c1;
(*cPtr).face = 1; // Modify a member using '.'
cPtr->face = 7;   // Modify a member using '->'
```

pointers

references

```
Card& cRef = c2;  // Reference to card
cRef.face = 13;
```

Exercise 1

- Write a function that prints the face and suit of a card in the following format:
  - *"Your card is a* **#** *of* **$***"*, where **#** is the face, and **$** is the suit of the card.
- Try with two choices of input:
  - A constant reference to a card object.
  - A pointer to a constant card object.

Exercise 2

- We have declared a deck of **52** cards as follows:

```
Card deck[52];
```

- Write a function that initializes this deck to **13** cards per suit. This function should also receive the array of C strings:

```
{ “Hearts”, “Diamonds”, “Clubs”, “Spades” }
```

- Write a function that shuffles the cards; that is, randomly swaps cards within the deck we declared above.

Exercise 3

- Find the error(s) and propose a solution!

(a)
```
Card *cPtr = &deck[2];
cout << *cPtr->face;
```

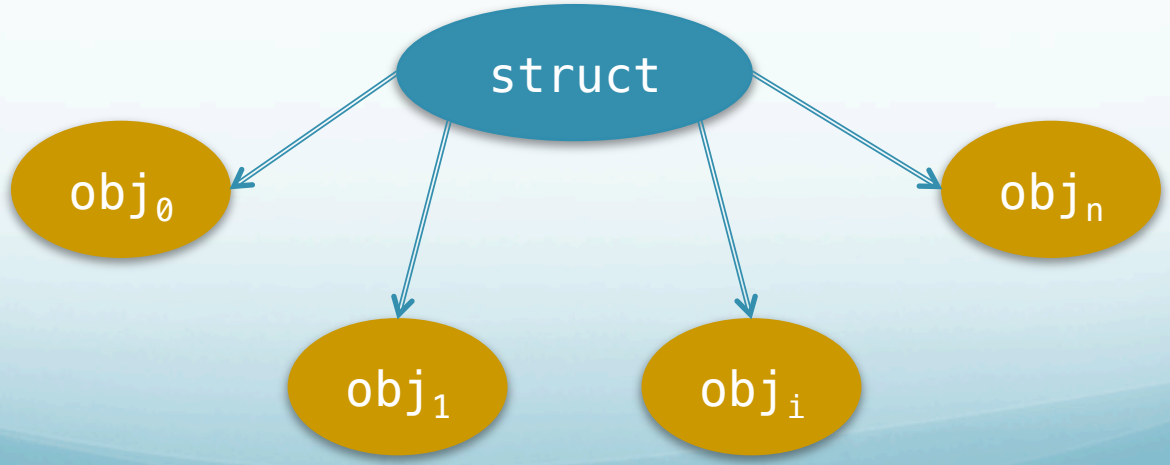
(b)
```
Card hearts[13];
hearts.suit = “Diamonds”;
```

(c)
```
struct Person {
   char lastName[15]
   char firstName[15]
   int age;
}
```

(d)
```
Person p;
Card d;
p = d;
```

Object Oriented Programming

- The OOP models real world objects through software. It **encapsulates** data (*attributes*) and functions (*behavior*) in packages called **objects**.
- Objects have the property of **hiding information**, but they can communicate with their surroundings (other objects) by well-defined **interfaces**.



OPP `struct` example

```
struct Time
{
private:
    int hour;     // 0 - 23.
    int minute;   // 0 - 59.
    int second;   // 0 - 59.

public:
    Time();
    void setTime( int, int, int );
    void print24();
    void printAMPM();
};
```

User-defined data type

Can be accessed only within member functions

data members

Can be accessed from outside the object

member functions

Defining Time's functions

```
// Constructor.
Time::Time()
{
        hour = minute = second = 0;
}

// Set time.
void Time::setTime( int h, int m, int s)
{
        hour = ( h >= 0 && h <= 23 )? h: 0;
        minute = ( m >= 0 && m <= 59 )? m: 0;
        second = ( s >= 0 && s <= 59 )? s: 0;
}
```

Scope operator `::`

How do we define?

```
void print24();
void printAMPM();
```

Creating Time's instances

```
Time sunset;                                    // Object
sunset.hour = 20;                 // Error!!!
sunset.minute = 5;                // Error!!!
sunset.second = 30;               // Error!!!
sunset.setTime( 20, 5, 30 );

Time *sunsetPtr = &sunset;                      // Pointer
(*sunsetPtr).print24();

Time& sunsetRef = sunset;                       // Reference
sunsetRef.printAMPM();

Time mealTimes[3];                              // Array
mealTimes[0].setTime( 11, 0, 0 );               //  Breakfast
(*(mealTimes+1)).setTime( 16, 30, 0 );          //  Lunch
(mealTimes+2)->setTime( 22, 30, 30 );           //  Dinner
```

Creating Time's const instances

- Is this code correct?

```
const Time midnight;
midnight.setTime( 0, 0, 0 );      // Error!!!
midnight.print24();               // Error!!!
midnight.printAMPM();             // Error!!!
```

- We need to fix the member functions' declaration and definition:

```
struct Time
{
private:
        int hour;       // 0 - 23.
        int minute;     // 0 - 59.
        int second;     // 0 - 59.

public:
        Time();
        void setTime( int, int, int );
        void print24() const;
        void printAMPM() const;
};
```

Exercise 4

- Write a member function to the `struct Time`, that receives another `Time` object and returns:
  - `0` if input object and receiver object are equal.
  - `+1` if receiver object is greater than input object.
  - `-1` if receiver object is smaller than input object.

```
int Time::compare( const Time& in ) const
```

Today's Material

- Find this material and the answers to programming exercises at:

  http://cs.ucla.edu/~langel/cs31/session8