Quiz.

What a mutator (like setRank) does to an object. Intro to ddd debugger to SEE VARIABLES and their CHANGES.

Watching Cards get initialized.

Randomizing the Pile by

- Picking a random Card.
- Swapping it with the Card where the random Card should be placed

Problems with scaling a random integer to an array index.
An object combines PROPERTIES (data, data members, member variables, components of STATE) with BEHAVIOR (methods, function methods, member functions) (The behavior is implemented by programmer written code that usually operates on the properties of the object.)
Class design for the Card class.
Properties: rank and suit.
Behavior:

▶ Set (CHANGE values of properties) its rank and suit.
▶ Report its rank and suit.
▶ Print its rank and suit AS SPECIFIED by Carrano.
▶ Report its value as an individual card in the valuing of bridge Hands.
class Card
{
    char rank;
    char suit;
    void setRank( char X );
};

void Card::setRank( char X )
{
    rank = X;
}

void someFunction()
{
    Card myCard;
    ...
    myCard.setRank( 'A' );
}
Review/FAST teaching of ARRAYS.

int X, Y, Z;
Card C1, C2, C3;

Card aC[5];

aC is FIVE variables, ALL THE SAME TYPE. WHICH of them to access is very conveniently determined AT RUNTIME by a computed integer value $i$, which must satisfy $0 \leq i \leq 4$.
aC[$i$] refers to ONE of the 5 Cards, depending on the value of $i$. 

```
aC
[0] [1] [2] [3] [4]
```
But, a Pile should allow us to program removing the Cards one-by-one so they can be dealt.
From Carrano page 155, adapted

const int MAX_LIST = 52; //Max length of a Pile
typedef Card ListItemType; //list item data type
ListItemType items[MAX_LIST]; //array of list items
int size; // LOGICAL length of the list

or more simply, for a Pile (of Cards)

const int MAX_LIST = 52; //Max length of a Pile
Card items[MAX_LIST]; //array of Cards
int size; // LOGICAL length of the list

Important principle: The part of the array that is actually used has subscripts in the range \([0...(\text{size} - 1)]\) (inclusive).
When \(\text{size} == 0\) NONE of the array is used
\([0...(\text{size} - 1)] = [0... - 1]\)
Let’s see this in action!
Why sort a Bridge Hand?

- Figure out how many Card are in each suit.
- Print the Cards in the order Bridge players like.

How does the way we randomized the Cards suggest an idea for a sorting algorithm?
Solving the Hand value computing problem

1. Use a value accumulation integer; initially 0.
2. Use **separate** functions or code sections for each rule!
3. Add the individual Card value for each Card (Aces count 4, etc.)
4. Voids (no card in a suit). For each SUIT, test if there are no Cards with it. If so, add 3 to the value. Repeat for the other suits.
5. Singletons (one Card in a suit). For each SUIT, test ... and add to value.
7. Detect and process long suits.
A Better Way, maybe

1. Sorting the cards brings the suits TOGETHER.
2. So, it’s easy to count the Cards of each suit.
3. Note the SAME function the number of points given the number of Cards applies for EACH of the four suits. (3 points for 0, 2 points for 1, 1 point for 6, 2 points for 7, etc.)
Tech info about the C/C++ library at UA ITSUNIX: Give the Unix command: man rand and get:

SYNOPSIS

```c
#include <stdlib.h>

int rand(void);
```

DESCRIPTION

The `rand()` function uses a multiplicative congruential random-number generator with period $2^{32}$ that returns successive pseudo-random numbers in the range of 0 to `RAND_MAX` (defined in `<stdlib.h>`).
This is our Array.

```
[0] [1] [2] [3]   [47] [48] [49] [50] [51]
```

This is the math. number line

```
0 1 2 3
```

0 <= int ret. by rand() <= RAND_MAX

Problem: program a formula to scale every integer between 0 and RAND_MAX inclusively to a real number and round down to get an integer between 0 and 51 inclusively.
This is our Array.

\[ \begin{array}{cccc}
0 & 1 & 2 & 3 \\
\end{array} \]

\[ 0 \leq \text{int } R=\text{rand()} \leq \text{RAND\_MAX} \]

\[ F=\text{static\_cast<double>}(R)/\text{RAND\_MAX}; \]

is a double float between 0.0 and 1.0

\[ SF = F\times52.0; \] is between 0.0 and 52.0

\[ \text{int } I=\text{static\_cast<int>}(SF); \] rounds down.
“Houston, we have a problem.”
In the unlikely case that \( R = \text{RAND\_MAX} \) exactly, I might be 52.
ARRAY OVERFLOW, CRASH!!
It’s a reasonable design choice to detect this case and “patch it,” I think.
Better solution

This is our Array.

0 1 2 3  [47][48][49][50][51]

This is the math. number line.

0 <= int R=rand() <= RAND_MAX

F=static_cast<double>(R)/(RAND_MAX+1);

is a double float between 0.0 & 1.0, and < 1.0

SF = F*52.0; is between 0.0 and 51.999999...

int l=static_cast<int>(SF); rounds down.
Alternative randomization algorithms:

- `rand() % mydeck.size()` (thanks, Evan K.)
- Do 200-300 random pairwise swaps (thanks, Andrew V.)
Randomizer implementation

```cpp
void Pile::randomize()
{
    int nUnplacedCards = 52;
    while( nUnplacedCards > 0 )
    {
        double F = static_cast<double>(Rand)/(RAND_MAX + 1);
        double SF = F*nUnplacedCards;
        int I = static_cast<int>( SF );
        /** @invariant nUnplacedCards == number of unplaced Cards.
           * Placed Cards are in items[nUnplacedCards...51].
           * When nUnplacedCards==52, there are no placed cards.
           */
        //The swapping pattern is applied below.
        Card tempC = items[ I ];
        items[ I ] = items[nUnplacedCards - 1];
        items[ nUnplacedCards - 1 ] = tempC;
        nUnplacedCards--;  //operation to RESTORE THE INVARIANT.
    }
}
```
Works on ITSUNIX, but CRASHED on Linux!!

seth@knowledge:~/CSI310/Lectures/L04/pr1$ ./PileTester
2C 3C 4C 5C 6C ..... 
Segmentation fault

here, I edited Pile::randomize(){} for debugging.

seth@knowledge:~/CSI310/Lectures/L04/pr1$ ./PileTester
2C 3C 4C 5C ....
Random from rand()=1804289383
RAND_MAX=2147483647
PileTester: Pile.cpp:55: void Pile::randomize():
Assertion ‘I >= 0 && I <= (nUnplacedCards - 1)’ failed.
Aborted

And the debugger showed that F was NEGATIVE!
“Albany, we have a problem.”
RAND_MAX on Linux was the LARGEST POSITIVE int, so (RAND_MAX + 1) OVERFLOWED.

The computer computed a NEGATIVE int!

Fix: Make sure RAND_MAX is converted to double float BEFORE adding 1.
What do we do?

int nUnplacedCards = 52;
while( nUnplacedCards > 0 )
{
    int Rand = rand(); // doc. says rand() returns int.

    #ifdef DEBUG
    cout << "Random from rand()=" << Rand << endl;
    cout << "RAND_MAX=" << RAND_MAX << endl;
    #endif

    Keep debugging code in place..

Code: ( static_cast<double>(RAND_MAX) + 1 )
This converts the int RAND_MAX to double precision floating point
type data with (approx.) the same numeric value.
That forces the “+ 1” calculation to be done with double floats.

double F = static_cast<double>(Rand)/
(static_cast<double>(RAND_MAX) + 1);

double SF = F*nUnplacedCards; //done in double

int I = static_cast<int>( SF );

assert( I >= 0 && I <= (nUnplacedCards - 1) );

/** @invariant checked by assert().
   * ( I >= 0 && I <= (nUnplacedCards - 1) )
   */
Code to Swap Cards and update an important variable

Remember, the NEXT unplaced card is in position (nUnplacedCards - 1).

```java
Card tempC;
tempC = items[ I ];
items[ I ] = items[nUnplacedCards - 1];
items[ nUnplacedCards - 1 ] = tempC;
nUnplacedCards--;
}
```