Pointers and Strings

Lecture 10

5.4 Calling Functions by Reference

- 3 ways to pass arguments to function
  - Pass-by-value
  - Pass-by-reference with reference arguments
  - Pass-by-reference with pointer arguments
- `return` can return one value from function
- Arguments passed to function using reference arguments
  - Modify original values of arguments
  - More than one value “returned”

5.4 Calling Functions by Reference

- Pass-by-reference with pointer arguments
  - Simulate pass-by-reference
    - Use pointers and indirection operator
  - Pass address of argument using **&** operator
  - Arrays not passed with **&** because array name already pointer
  - ***** operator used as alias/nickname for variable inside of function

fig05_06.cpp (1 of 2)

```
// Fig. 5.6: fig05_06.cpp
// Cube a variable using pass-by-value.
#include <iostream>

using std::cout;
using std::endl;

int cubeByValue( int );   // prototype

int main()
{
   int number = 5;

   cout << "The original value of number is " << number;

   // pass number by value to cubeByValue
   number = cubeByValue( number );

```

fig05_06.cpp
(2 of 2)

fig05_06.cpp
output (1 of 1)

```
  cout << "\nThe new value of number is " << number << endl;

  return 0;  // indicates successful termination

} // end main

// calculate and return cube of integer argument
int cubeByValue( int n )
{
  return n * n * n; // cube local variable n and return result

} // end function cubeByValue
```

```
The original value of number is 5
The new value of number is 125
```


fig05_07.cpp
(1 of 2)

```
// Fig. 5.7: fig05_07.cpp
// Cube a variable using pass-by-reference
// with a pointer argument.
#include <iostream>

using std::cout;
using std::endl;

void cubeByReference( int * );   // prototype

int main()
{
  int number = 5;

  cout << "The original value of number is " << number;

  // pass address of number to cubeByReference
  cubeByReference( &number );
```

```
19
20   cout << "\nThe new value of number is " << number << endl;
21
22   return 0;  // indicates successful termination
23
24 } // end main
25
26 // calculate cube of *nPtr; modifies variable number in main
27 void cubeByReference( int *nPtr )
28 {
29   *nPtr = *nPtr * *nPtr * *nPtr;  // cube *nPtr
30
31 } // end function cubeByReference
```

fig05_07.cpp (2 of 2)

fig05_07.cpp output (1 of 1)

```
int value = *nPtr;
*nPtr = value * value * value;
```

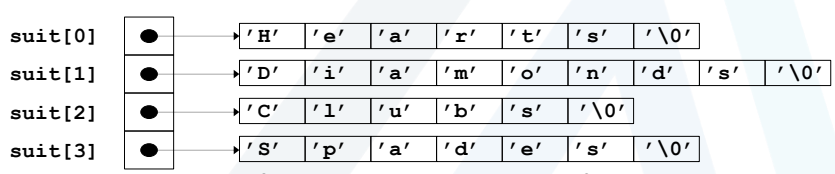
```
The original value of number is 5
The new value of number is 125
```


5.9 Arrays of Pointers

- Arrays can contain pointers
  - Commonly used to store array of strings

```
char *suit[ 4 ] = {"Hearts", "Diamonds",
                   "Clubs", "Spades" };
```

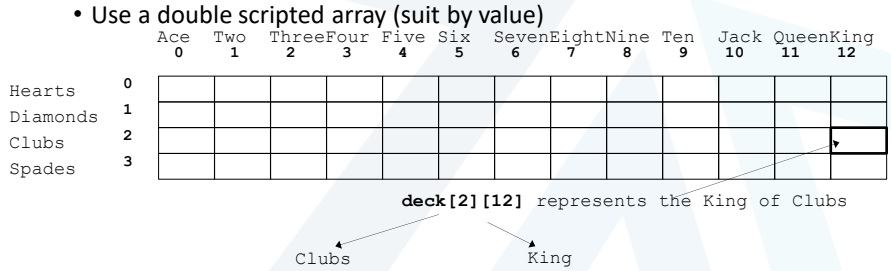
  - Each element of **suit** points to `char *` (a string)
  - Array does not store strings, only pointers to strings



  - `suit` array has fixed size, but strings can be of any size

5.10 Case Study: Card Shuffling and Dealing Simulation

- Card shuffling program
  - Use an array of pointers to strings, to store suit names
  - Use a double scripted array (suit by value)



  - Place 1-52 into the array to specify the order in which the cards are dealt

5.10 Case Study: Card Shuffling and Dealing Simulation

- Pseudocode for shuffling and dealing simulation

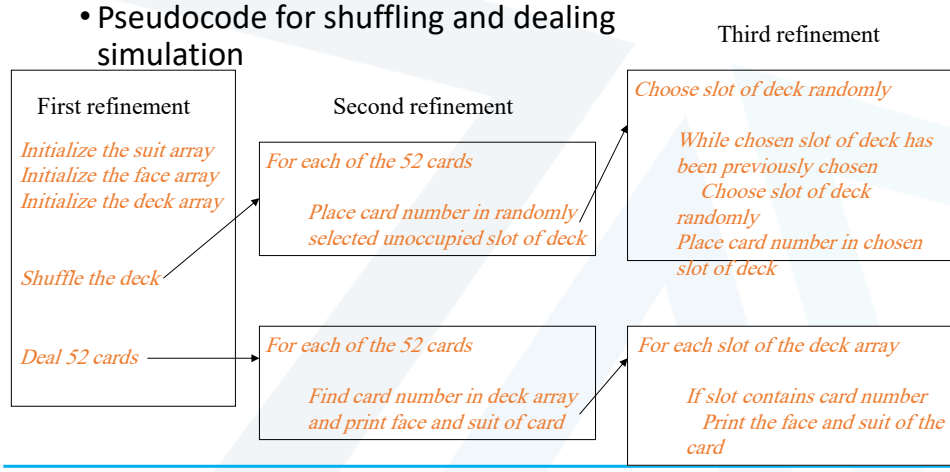


fig05_24.cpp (1 of 4)

```
// Fig. 5.24: fig05_24.cpp
// Card shuffling dealing program.
#include <iostream>

using std::cout;
using std::left;
using std::right;

#include <iomanip>

using std::setw;

#include <cstdlib>  // prototypes for rand and srand
#include <ctime>    // prototype for time

// prototypes
void shuffle( int [][ 13 ] );
void deal( const int [][ 13 ], const char *[], const char *[] );

struct Card{
  const char * suit;
  const char * rank;
  Card () {suit = NULL; rank = NULL;}
};
```


fig05_24.cpp (2 of 4)

```
const char *suit[ 4 ] =
  { "Hearts", "Diamonds", "Clubs", "Spades" };

 // initialize face array
  const char *face[ 13 ] =  { "Ace", "Deuce", "Three", "Four", "Five",
      "Six", "Seven", "Eight",  "Nine", "Ten", "Jack", "Queen", "King" };
int main()   {
  // initialize suit array

  // initialize deck array
  Card deck[ 52 ];

  srand( time( 0 ) );       // seed random number generator

  shuffle( deck );
  deal( deck, face, suit );
  return 0;  // indicates successful termination
} // end main
```

fig05_24.cpp (3 of 4)

```
// shuffle cards in deck
void shuffle( Card wDeck[])   {
  int row;
  int column;

  // for each of the 52 cards, choose slot of deck randomly
  for ( int card = 1; card <= 52; card++ )  {
    // choose new random location until unoccupied slot found
    do {
      row = rand() % 4;
      column = rand() % 13;
    } while ( wDeck.suit != NULL ); // end do/while
    // place card number in chosen slot of deck
    wDeck[ card ]. = card;
    wDeck[ row ][ column ] = card;

  } // end for

} // end function shuffle
```


fig05_24.cpp (4 of 4)

```
// deal cards in deck
void deal( const int wDeck[][ 13 ], const char *wFace[],
        const char *wSuit[] )   {
  // for each of the 52 cards
  for ( int card = 1; card <= 52; card++ )
    // loop through rows of wDeck
    for ( int row = 0; row <= 3; row++ )

      // loop through columns of wDeck for current row
      for ( int column = 0; column <= 12; column++ )

        // if slot contains current card, display card
        if ( wDeck[ row ][ column ] == card ) {
          cout << setw( 5 ) << right << wFace[ column ]
              << " of " << setw( 8 ) << left
              << wSuit[ row ]
              << ( card % 2 == 0 ? '\n' : '\t' );
        } // end if
} // end function deal
```

Cause face to be output right justified in field of 5 characters.

Cause suit to be output left justified in field of 8 characters.

fig05_24.cpp output (1 of 1)

```
 Nine of Spades          Seven of Clubs
 Five of Spades          Eight of Clubs
Queen of Diamonds        Three of Hearts
 Jack of Spades           Five of Diamonds
 Jack of Diamonds        Three of Diamonds
Three of Clubs             Six of Clubs
  Ten of Clubs            Nine of Diamonds
  Ace of Hearts          Queen of Hearts
Seven of Spades          Deuce of Spades
  Six of Hearts          Deuce of Clubs
  Ace of Clubs           Deuce of Diamonds
 Nine of Hearts          Seven of Diamonds
  Six of Spades          Eight of Diamonds
  Ten of Spades           King of Hearts
 Four of Clubs             Ace of Spades
  Ten of Hearts           Four of Spades
Eight of Hearts          Eight of Spades
 Jack of Hearts            Ten of Diamonds
 Four of Diamonds         King of Diamonds
Seven of Hearts           King of Spades
Queen of Spades           Four of Hearts
 Nine of Clubs             Six of Diamonds
Deuce of Hearts           Jack of Clubs
 King of Clubs           Three of Spades
Queen of Clubs            Five of Clubs
 Five of Hearts            Ace of Diamonds
```


fig05_24.cpp (1 of 4)

```cpp
// Fig. 5.24: fig05_24.cpp
// Card shuffling dealing program.
#include <iostream>

using std::cout;
using std::left;
using std::right;

#include <iomanip>

using std::setw;

#include <cstdlib>  // prototypes for rand and srand
#include <ctime>    // prototype for time

// prototypes
void shuffle( int [][ 13 ] );
void deal( const int [][ 13 ], const char *[], const char *[] );

```

fig05_24.cpp
(2 of 4)

```
int main()   {
   // initialize suit array
   const char *suit[ 4 ] =
      { "Hearts", "Diamonds", "Clubs", "Spades" };

   // initialize face array
   const char *face[ 13 ] =  { "Ace", "Deuce", "Three", "Four", "Five",
      "Six", "Seven", "Eight",  "Nine", "Ten", "Jack", "Queen", "King" };

   // initialize deck array
   int deck[ 4 ][ 13 ] = { 0 };

   srand( time( 0 ) );       // seed random number generator

   shuffle( deck );
   deal( deck, face, suit );
   return 0;  // indicates successful termination
} // end main

```


fig05_24.cpp
(3 of 4)

```
// shuffle cards in deck
void shuffle( int wDeck[][ 13 ] )   {
   int row;
   int column;

   // for each of the 52 cards, choose slot of deck randomly
   for ( int card = 1; card <= 52; card++ )  {
      // choose new random location until unoccupied slot found
      do {
         row = rand() % 4;
         column = rand() % 13;
      } while ( wDeck[ row ][ column ] != 0 ); // end do/while
      // place card number in chosen slot of deck
      wDeck[ row ][ column ] = card;

   } // end for

} // end function shuffle

```

fig05_24.cpp (4 of 4)

```
// deal cards in deck
void deal( const int wDeck[][ 13 ], const char *wFace[],
        const char *wSuit[] )   {
  // for each of the 52 cards
  for ( int card = 1; card <= 52; card++ )
    // loop through rows of wDeck
    for ( int row = 0; row <= 3; row++ )

      // loop through columns of wDeck for current row
      for ( int column = 0; column <= 12; column++ )

        // if slot contains current card, display card
        if ( wDeck[ row ][ column ] == card ) {
          cout << setw( 5 ) << right << wFace[ column ]
              << " of " << setw( 8 ) << left
              << wSuit[ row ]
              << ( card % 2 == 0 ? '\n' : '\t' );
        } // end if
} // end function deal
```


fig05_24.cpp output (1 of 1)

```
 Nine of Spades          Seven of Clubs
 Five of Spades          Eight of Clubs
Queen of Diamonds        Three of Hearts
 Jack of Spades           Five of Diamonds
 Jack of Diamonds        Three of Diamonds
Three of Clubs             Six of Clubs
  Ten of Clubs            Nine of Diamonds
  Ace of Hearts          Queen of Hearts
Seven of Spades          Deuce of Spades
  Six of Hearts          Deuce of Clubs
  Ace of Clubs           Deuce of Diamonds
 Nine of Hearts          Seven of Diamonds
  Six of Spades          Eight of Diamonds
  Ten of Spades           King of Hearts
 Four of Clubs             Ace of Spades
  Ten of Hearts           Four of Spades
Eight of Hearts          Eight of Spades
 Jack of Hearts            Ten of Diamonds
 Four of Diamonds         King of Diamonds
Seven of Hearts           King of Spades
Queen of Spades           Four of Hearts
 Nine of Clubs             Six of Diamonds
Deuce of Hearts           Jack of Clubs
 King of Clubs           Three of Spades
Queen of Clubs            Five of Clubs
 Five of Hearts            Ace of Diamonds
```

5.12.1 Fundamentals of Characters and Strings

- Character constant
  - Integer value represented as character in single quotes
  - `'z'` is integer value of `z`
    - `122` in ASCII
- String
  - Series of characters treated as single unit
  - Can include letters, digits, special characters `+`, `-`, `*` ...
  - String literal (string constants)
    - Enclosed in double quotes, for example:

      `"I like C++"`
  - Array of characters, ends with null character `'\0'`
  - String is constant pointer
    - Pointer to string's first character
      - Like arrays

5.12.1 Fundamentals of Characters and Strings

- String assignment
  - Character array
    - `char color[] = "blue";`
      - Creates 5 element `char` array `color`
        - last element is `'\0'`
  - Variable of type `char *`
    - `char *colorPtr = "blue";`
      - Creates pointer `colorPtr` to letter `b` in string "`blue`"
        - "`blue`" somewhere in memory
  - Alternative for character array
    - `char color[] = { 'b', 'l', 'u', 'e', '\0' };`

5.12.1 Fundamentals of Characters and Strings

- Reading strings
  - Assign input to character array **`word[ 20 ]`**

    `cin >> word`
    - Reads characters until whitespace or EOF
    - String could exceed array size

      `cin >> setw( 20 ) >> word;`
    - Reads 19 characters (space reserved for **`'\0'`**)

5.12.1 Fundamentals of Characters and Strings

- **`cin.getline`**
  - Read line of text
  - **`cin.getline( array, size, delimiter );`**
  - Copies input into specified **`array`** until either
    - One less than **`size`** is reached
    - **`delimiter`** character is input
  - Example

```
char sentence[ 80 ];
cin.getline( sentence, 80, '\n' );
```

or

```
string sent;
getline(cin, sent, '\n');
```