# Lecture 17: Vectors

PIC 10A

Todd Wittman


## Drawing Cards

- When we talked about random numbers, we mentioned using a switch statement to pick a random card suit.

```
string suit;
int suit_number = 1+rand()%4;
switch(suit_number) {
   case 1:  suit = "Clubs";  break;
   case 2:  suit = "Diamonds";  break;
   case 3:  suit = "Hearts";  break;
   case 4:  suit = "Spades";  break;
}
```


- I'll let you figure out how to pick a random rank.
- Can we use this to make a Card class?

## The Card Class

- We could construct a Card class that stores a playing card.

```
class Card {
   public:
        Card( );
        ....
   private:
        string suit;
        string rank;
};
```

What other member functions should we have?

- It would be cool if a call to our constructor (Card card1;) drew a random card.

## The Card Class

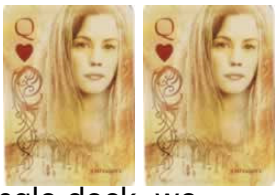
- It would be nice if the default constructor for a Card class generated a random card.

```
Card::Card( ) {
   string suit;
   int my_rand = 1+rand()%4;
   switch(my_rand) {
        case 1:  suit = "Clubs";  break;
        case 2:  suit = "Diamonds";  break;
        case 3:  suit = "Hearts";  break;
        case 4:  suit = "Spades";  break;
   }
   ....
```

We already have suit as a private variable. Don't re-declare it!

## Simulate Drawing Cards

- In next week's HW, you can assume the cards are drawn from more than one deck.



- So it's OK to get

  card1 = "Queen of Hearts"

  card2 = "Queen of Hearts"

- To simulate drawing 2 cards from a single deck, we could "redraw" if we get the same card.

```
Card card1;   Card card2;
while (card1 == card2)
        Card card2;
```

- But this assumes == is defined for cards.
- How would you redraw using just the < operator?

## Storing Data in Lists

- Suppose we want to keep track of a list 5 numbers.

  11 -12 42 38 -105

- We could create 5 int variables.

```
int value1, value2, value3, value4, value5;
```

- But this is a lot of work and doesn't scale well.
- What if we have a list of 10 numbers? 100?
- How would we manage a database or an image?
- C++ gives us a way to store a list of numbers in a single variable.
- Without this feature, real computer programming would be impossible.

## Arrays vs. Vectors

- In Chapter 6, your book describes two options for storing data in lists: arrays and vectors.
- Arrays are the old-fashioned, simple way to store lists. Arrays are built-in primitive data types in C++. Most C++ code out there uses arrays.
- Vectors are the newer, fancier version of arrays with some special functions added on. Vectors are defined in the <vector> standard library.
- The usage for both types is very similar, with subtle differences.
- We'll start vectors today and cover arrays next week.

## Sec 6.1: Declaring Vectors

- The vector class allows us to store a list of objects of the same type.
- To use vectors, we have to #include <vector>
- To declare a vector, use the syntax

  vector<data_type> variable_name (size);
- Omitting the number size makes it an empty size 0 list.
- To create a vector of 10 integers, we write

  vector<int> my_vector(10);
- We can use any data type, even classes.

  vector<string> word_list(200);
- We can even use our own user-defined classes.

  vector<Card> deck(52);

## Sec 6.2: Accessing Vectors

- To access the element at position i of a vector, we use brackets [i].

```
my_vector [3] = 4.2;
```

- As with strings, vector indices run 0 to SIZE-1.
- Similar to the string length() function, vectors come equipped with a size( ) member function.

```
for (int i = 0; i < my_vector.size( ); i++)
    cout << my_vector [i] << "\n";
```

## 3 Simple Examples

- We often use for loops to assign vector values.
- Ex Store the first 100 even integers.

```
vector<int> evens(100);
for (int i = 0; i<100; i++)
        evens[i] = 2*i+2;
```

- Ex Read in 10 doubles and compute the sum.

```
vector<double> list(10);
double sum = 0;
for (int i = 0; i < 10; i++)  {
        cin >> list[i];
        sum += list[i];
}
cout << "\nThe sum is: " << sum;
```

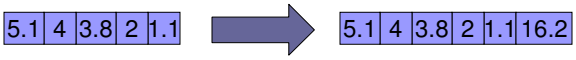
- Ex Read in 4 hobbit names.

```
vector<string> hobbits(4);
for (int i = 0; i < 4; i++)  {
        cout << "Enter the name of hobbit #" << i+1 <<": ";
        cin >> hobbits[i];
}
```

## push_back / pop_back

- We can change the size of vectors.
- To add one more element to the back end of a vector with a given *value*, use the push_back(*value*) function.

```
vector<double> my_vector(5);
my_vector.push_back(16.2);     //my_vector now has size 6.
```




- To remove one element from the end, use the pop_back( ) function.

```
vector<double> my_vector(5);
my_vector.pop_back( );       //my_vector now has size 4.
```


## Vectors are Dynamic

- Suppose we are reading in a list of integers from the user, but we don't know how many they'll type in.
- We can start with an empty vector and just push_back numbers until we get a cin failure.

```
vector<int> numbers;    //Size 0 vector.
int entry;
while (cin >> entry)
        numbers.push_back(entry);
```

- Now numbers stores however many numbers the user typed in. If we want to know how many that is, we just look up numbers.size().

## The resize function

- The resize member function resets the vector to the new size.

```
vector<int> v(10);      //v starts with 10 elements
v.resize(20);      //Now v has 20 elements
```

- If the new size > old size, it adds blank elements.
- If the new size < old size, it deletes off the back end.

```
v.resize(5);    //We kept just the first 5 elements of v
```

- Sometimes we want to start over and go back to an empty vector. The easy way is to resize it back to 0.

```
v.resize(0);    //Now v is an empty vector.
```

## The Hand Class

- In card games, each player has a set of cards (hand).
- We can build a Hand class on top of our Card class.

```
class Hand {
   public:
        Hand();
        ...
   private:
        vector<Card> cards;    //The list of our cards.
};
```

- What functions should this class have?

## The Hand Class

- Every time the player draws a card, we should add it to the vector cards in our Hand.

```
void Hand::drawCard() {
        Card newCard;   //Creates a random card.
        cards.push_back(newCard);
        return;
}
```

- When a round of play is over, the player turns in all her cards.

```
void Hand::resetHand() {
        cards.resize(0);
        return;
}
```

## Blackjack

- In the game Blackjack (21), a player plays against the dealer.
- Whoever gets the sum of their card values closest to 21 without going over, wins.
- In a casino, the dealer draws cards until the sum is >= 17.
- Suppose we had a sum() member function in our Hand class that adds up the card values.

```
Hand dealer;
while (dealer.sum() < 17)
    dealer.drawCard();
```

- Once the round of play is over, the dealer turns in her cards.

```
dealer.resetHand();
```

## Sec 6.3: Vectors in Functions

- To pass a vector to a function, use the prototype

```
void function_name (vector<type> vector_name) { ...
```

- Note we have to tell it what type the vector holds.
- It is not necessary to tell the function the size of the vector, because it can look it up with the size() function.

```
void print (vector<int> list) {
        for (int i = 0; i < list.size( ); i++)
                cout << list [i] << " ";
        return;
}
```

## Vectors in Functions

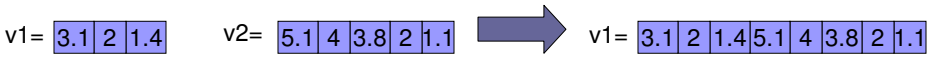
- A function can return a vector.
- Ex Extract the double sub-vector between the start and finish position of a given vector.


```
vector<double> extract (vector<double> v, int start, int finish) {
        vector<double> sub (finish-start+1);
        for (int i = start; i <= finish; i++)
                sub[i-start] = v [i];
        return sub;
}
```

Value parameter. Changes to v are not sent back.

# Appending Vectors

- Ex Write a function which appends double vector v2 onto the back end of vector v1.




Pass by reference

Pass by value

```
void append (vector<double>& v1, vector<double> v2)   {
        for (int i = 0; i < v2.size( ); i++)
                v1.push_back ( v2[i] );
        return;
}
```