Lecture 18: Vectors in Functions

PIC 10A
Todd Wittman

Quick Review of Vectors

- A vector is a class defined in the <vector> library that can store a list of data.
- We decide the data type and initial size in the declaration of the vector.

```
vector<string> words(10);    vector<int> list(200);
```

- We can look up the current size of a vector.

```
int number_words = words.size();
```

- To access an element, use brackets [].
- Remember vector indices start at 0.

```
words[0]="hello";      cout<<list[1]+list[2];
```

- Vectors are dynamic: they can change size.

```
words.resize(5);        //Keep just first 5 elements.
words.push_back("Frodo");    //Add "Frodo" to end.
words.pop_back();    //Remove last element.
```

Vector Member Functions

vector<T> -- T is the data type (e.g. int, double, string, Card, etc.)

vector (int n)	Constructs a vector with n elements.
int size( )	Returns current size of vector.
push_back (T x)	Inserts x at the back end of the vector.
pop_back( )	Removes the last element from the back end of the vector.
resize (int n)	Resizes vector to size n. If n is smaller than old size, deletes elements at back end.

Strings as Vectors

- A string is actually vector<char>

```
string name = "Frodo";
name[0] = 'G';
cout << name;        // Prints Grodo
```

- A string variable can use vector member functions.

```
name.push_back('s');    //Adds one char to the end.
```

- Recall On Practice Exam 1, you had to replace every s in a string with an f.
- It was a hard problem because you had to extract the single letter substring, then delete the "s", and then insert the "f".
- Much easier with vector notation.

```
for (int i = 0; i < my_string.length(); i++)  {
    if (my_string[i] == 's')
        my_string[i] = 'f';
}
```

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.

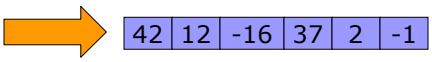
```
void print (vector<int> list) {
        for (int i = 0; i < list.size( ); i++)
                cout << list [i] << "\n";
        return;
}
```

- Note that cout<<list would not print the list.

Reading In Data

- Functions can return vectors.
- Ex Read in a list of integers from the user and put it into a vector. Stop when they enter a non-integer.
- Recall that (cin>>x) returns a boolean: true if it read x successfully, false if input fails.

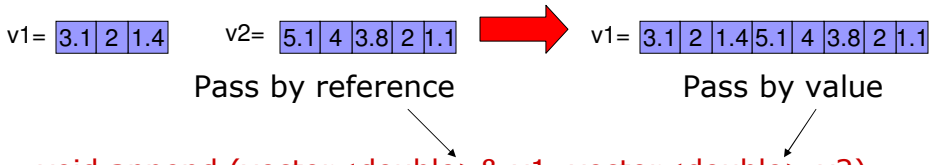
```
42   12 -16
37 2     -1   frodo
```




```
vector<int> read () {
   vector<int> numbers;
   int entry;
   while (cin >> entry)
        numbers.push_back(entry);
   return numbers;
}
```

Appending Vectors

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




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

- What would change if we passed v2 by reference?

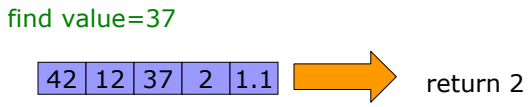
Conserving Memory

- Every time a variable is passed by value to a function, a local copy of the variable is created.
- So passing a vector storing 1,000,000 integers by value would create a local vector of size 1,000,000 just for that function.
- This will eat up memory and potentially crash our program.
- Many programmers suggest always passing a vector by reference. Just remember that changes will be recorded, so be careful.
- One way to prevent accidental changes is to use the const modifier inside the parameters.

```
void my_function (const vector<int>& my_vector)
```

Searching a List

- Ex Search through a vector for a specific value and return the position of its first occurrence.

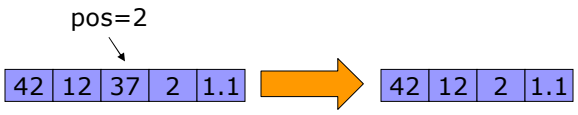



```
int findX (const vector<int>& v, int value) {
   int pos = 0;
   while (pos<v.size() && v[pos] != value)
        pos++;
   return pos;
}
```

- What does this return if x was not in v?

Erasing an Element

- Ex Erase an element from a double vector v at a given position: erase(v,2);

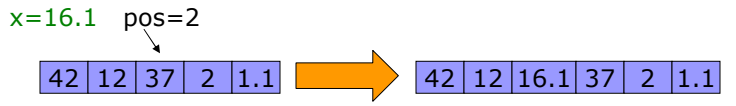


Note pass by reference so we record changes.

```
void erase(vector<double>& v, int pos) {
   for (int i = pos; i < v.size() - 1; i++)
        v[i] = v[i+1];
   v.pop_back();
}
```

Inserting an Element

- Ex Insert a double x before a given position in a double vector v: insert(v,2,x);



Note pass by reference so we record changes.

```
void insert(vector<double>& v, int pos, double x) {
  int last = v.size() - 1;
  v.push_back(v[last]);
  for (int i = last; i > pos; i--)
      v[i] = v[i - 1];
  v[pos] = x;
}
```

Sec 6.4: Parallel Vectors

- Suppose we want to keep track of a product names, prices, and score on 1-10 scale.
- We should keep 3 *parallel vectors*: name, price, and score.
- Any changes to one vector should also be made to the others, so that name[i], price[i], and score[i] all correspond.

```
vector<string> name(6);
vector<double> price(6);
vector<int> score(6);
erase(name,3);
erase(price, 3);
erase(score,3);
```

	Name	Price	Score
0	Wii	249.99	8
1	PC	1299	5
2	Mac	800	6
3	PS3	600	3
4	XBox360	499.49	7
5	Atari2600	2.99	10

A Better Solution

- This is kind of annoying to keep track of 3 separate vectors.
- A better solution is to create a class Product with private variables name, price, and score.
- Then just create a vector of Products.

```
class Product {
    ....
private:
    string name;
    double price;
    int score;
};
```

```
vector<Product> list(6);
erase(list,3);
```

Sorting Integers

- There is a vector/array sorting function in the <algorithm> library.
- The sorting function looks like this:

sort (*pointer to beginning of vector, pointer to end of vector*);

- The vector class has member functions begin() and end() which return pointers to vector elements.
- For example, the code below sorts and prints a vector of integers.

```
#include <algorithm>
        ...
vector<int> v(4);
        v[0]=42;        v[1]=12;        v[2]=37;        v[3]=2;
        sort( v.begin(), v.end() );
        for (int i = 0; i < v.size(); i++)
                cout << v[i] << " ";
```

OUTPUT: 2 12 37 42

Sorting Other Data Types

- We can also sort doubles.

42.1	27.2	-2.0	-2.1

→

-2.1	-2.0	27.2	42.1

- We can sort strings too, but remember lexicographic order is strange.

frodo	FRODO	223	aardvark

→

223	FRODO	aardvark	frodo

- Can we sort our Player class? No, because the compiler doesn't know how to compare two player.

  player1 < player2 ???

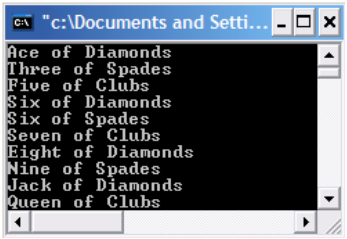
- Can we sort our Card class?
- Yes, because you defined the operator< .

  card1 < card2

Sorting A Deck of Cards

- Create and sort a deck of 52 cards. *(Not for HW.)*

```
vector<Card> deck(52);
for (int i=0; i < deck.size(); i++) {
   Card newcard;
   deck[i] = newcard;
}
sort (deck.begin(), deck.end());
for (int i=0; i < deck.size(); i++)
   cout << deck[i].get_rank() << " of "
          << deck[i].get_suit() << "\n";
```



- Note this sorts on the rank, but the suits are in random order.

Shuffle A Deck of Cards

- Ex Shuffle a deck of 52 cards.
- We want to call a function shuffle like below:

```
vector<Card> deck(52);
shuffle(deck);
```

- Idea: Pick 2 random cards from the deck and swap their positions. Repeat this many times.
- Let's do 1000 swaps.

Shuffle A Deck of Cards

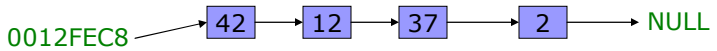
```
void shuffle (vector<Card>& deck) {
   int pos1, pos2;
   Card temp;
   for (int i = 1; i <= 1000; i++) {
        pos1 = rand()%52;   //Random # 0-51.
        pos2 = rand()%52;   //Another random #0-51.
        temp = deck[pos1];
        deck[pos1] = deck[pos2];
        deck[pos2] = temp;
   }
   return;
}
```

What does this print out?

- What happens when we try to print out the array variable name?

```
int v[4];
v[0]=42;   v[1]=12;   v[2]=37;   v[3]=2;
cout << v;
OUTPUT:  0012FEC8
```

- This is the hexadecimal memory address of the first element of v.
- The elements of the array store a value and a <u>pointer</u> to the location of the next element.



2D Vectors

- We will see Wednesday creating a 2D array is easy.

```
int my_array[3][5];   //Creates 3x5 matrix.
```

- Creating it with vectors is not so easy.
- <u>Method1</u>: Create a list of size rows*cols and keep track of rows yourself.

```
vector<int> my_vector(15);      //List of length 15
```

- <u>Method2</u>: Create a vector of vectors.

```
vector<vector<int>> my_vector(3);   //3 rows
vector<int> row(5);    //Row with 5 columns.
for (int i = 0; i < 3; i++)
    my_vector[i] = row;    //Creates 3x5 matrix.
```

- Probably easier to use arrays for 2D data.