CS 102 Introduction to Programming Using C++

Chapter 6

Arrays and Vectors

Homework

- Written homework
- R6.1, 2, 6, 10, 11, 13 to 18, 23, 25

- Programs
- p. 295, choose one of P6.1, 2, 9
    - If you choose P6.2, choose four parts
- Also choose one of P6.12, 14

Passing an Array into a Function

- There are several important things to know if you want to pass an array into a function
- There are some important differences between passing an array and passing a simple variable
- Some differences are syntax differences
- Some differences involve knowing about what happens when using an array inside a program

Passing an Array into a Function

- You are writing a function to add up the values in an array
- For example,

```
int sum (int int_array[])
```

- you have to add the brackets to indicate that a parameter is an array
  - We don't specify the size inside the brackets
- You also need to add another parameter to tell the size of the array
- So, the actual call should be

```
int sum (int int_array[], int array_size)
```

Using Size As a Parameter

- As before, only a portion of the array might be in use
- We will use an int (like number_of_scores) to keep track of the number of elements in use
- We could then pass that into a function
- The function might add or delete elements from the array
    - This means it adjusts the int

Changing the Effective Size of an Array inside a Function

- The function will have to alert the caller that the size has changed
- A common way to do that is to return the new size
- This function would then be an int function

Capacity vs. Size

- We need to keep track of the number of elements in use in an array
- We also need to keep track of the capacity of the array
  - This is the initial size of the array
  - We might have declared the array

        int counts [10]

    - Here, 10 is the capacity
- This might have to be passed to the function
  - We definitely need to pass the capacity if the function might add new elements to the array

Array Parameters Are Reference Parameters

- Array parameters are always reference parameters
    - If they were value parameters, a copy of the array would be passed into the function
    - Making this copy would take too long
- This means array data is “live” within a function
- Unlike variables, any changes you make to an array while inside a function are actually made to the array
- Also, unlike variables, you do not need (and we don’t use) the & on an array

Returning an Array from a Function

- Functions cannot return arrays
  - In fact, they can only return simple things, like variables
- That should not be a problem
  - If an array is a parameter, it's a reference parameter and can be changed

Two-Dimensional Arrays

- We can create two-dimensional arrays
- You use the idea of a two-dimensional array when you attend a sporting event or a concert in a concert hall
- A matrix in math is a two-dimensional array
- Two-dimensional arrays are useful in many contexts

Two-Dimensional Arrays

- Think of a two-dimensional array as having rows and columns
- To create a two-dimensional array, you declare it

```
            int concert_hall [number_of_rows]  [number_of_columns];
```

- Notice that you need two pairs of brackets
- You can also create and initialize a two-dimensional array

```
            int matrix [2] [3]= {
                                 {1, 2, 3},
                                { 7, 9, 2}
                              }
```

- You do have to specify the dimensions
- We do line the rows up like that

Using a Two-Dimensional Array

- Using an element of a two-dimensional array is no different than using a variable or an element of a one dimensional array
- For example, you can add up all the elements in the first row of an a two-dimensional array

Using a Two-Dimensional Array Examples 1 and 2

- To add all elements in the first row

```
int sum = 0;
int row_num = 0;
  for (int i=0;  i<num_cols;  i++)
      sum += nums [row_num] [i];
```

- To add all elements in the first column

```
int sum = 0;
   int col_num = 0;
     for (int i=0;  i<num_rows;  i++)
         sum += nums [i] [col_num];
```

Functions and Two-Dimensional Arrays

- You have to be careful when writing a function that will use a two-dimensional array
- You have to specify the number of columns in the function header
- For example:
  - In main ()

    ```
    sales_reps [5] [3];
    ```
  - In the function definition

    ```
    int add_reps (arr [] [], which_row)
    ```
- This is not OK. It will generate a syntax error.

The Correct Way to Call a Function That Uses Two-dimensional Arrays

- If in main () you have

```
sales_reps [5] [3];
```

- Then in the function definition you will need

```
int add_reps (arr [] [3], which_col)
```

- The number of columns must be specified
- This is because of how the array is stored in memory

And Now for Something New: Vectors

- A vector is similar to an array
- It can grow and shrink as needed
  - Because of this, it is more flexible than an array
- To use it, you need to add

  ```
  #include <vector>
  ```
- When declaring a vector, you have to use this syntax

  ```
  vector <int> nums (10);
  ```
  - This creates a vector that can contain 10 integers
  - Notice that this looks different from declaring an int array

A Vector Can Be Treated Just Like an Array

- You can have this code in a program

```
vector <int> nums (10);
for (int i=0;  i<10; i++)
        cin >> nums [i];
int sum  = 0;
for (int i=0;  i<10;  i++)
    sum += nums [i];
```

Adding New Elements to a Vector

- You can add items to a vector
  - You use the push_back function
  - The added element goes to the end of the vector

```
vector <int> nums (10);
for (int i=0;  i<10; i++)
    cin >> nums [i];
int sum  = 0;
for (int i=0;  i<10;  i++)
    sum += nums [i];
nums.push_back (1000);
```

  - The vector now has 11 items!

Adding Elements to an Empty Vector

- It’s common to create an empty vector and add elements to it using push_back()

```
vector <int> nums;
while (cin >> input)
        nums.push_back (input);
```

Removing Elements from a Vector

- There is a pop_back () function
- It is similar to the push_back () function
- It removes the last element from a vector

Where Are We Now?

- You create a vector
- You add some elements
- You remove some elements
- You add some more elements
- You lose track of how many elements are in the vector
- What can you do?

Determining the Size of a Vector

- No worries. There is a size function

```
vector <int> nums;
/*  Deleted lines
     with many push_back and pop_back calls  */
for (int i=0;  i<nums.size();  i++)
    cout << nums [i];
```

Using Vectors

- You can use a vector element anywhere you can use any variable
- You can print a vector element

```
cout << nums [4];
```

- You can use it in a calculation

```
answer = 3*nums [2] + 7;
nums [6] = 12 + 16*i;
```

- You can use it as an argument to a function

```
add_1_to_value (nums [5]);
```

Vectors vs. Arrays

- Vectors are mostly like resize-able arrays
- There is one important difference
- If you pass an entire vector to a function, it is passed by value
  - You cannot change it in the function
- If you want to change it, you must pass it by reference

```
new_function (vector<int>& nums)
```

A Function Can Return a Vector!

- Here is a function that reads numbers into a vector and returns the vector

```
vector <int> read_nums ()
{
   vector <int> temp_nums;
   int input;
   while (cin >> input)
       temp_nums.push_back (input);
   return temp_nums;
}
```

Copying a Vector to a Vector

- It is very easy to copy one vector to another
- You simply copy it like you copy two ints

```
vector <double> numbers;
//  Lots of  deleted lines
vector <double> copy_of_numbers;
copy_of_numbers = numbers;
```

Finding Values That Meet Criteria

- Suppose customer numbers are always in the range 1000-9999 (four digits long)
- You have an array cust_nums [100]
  - It contains customer numbers
- You would like to validate this array
- One thing you can do is to keep track of all offending values in the array
- You could use this code

Array Code to Find Values

```
int cust_nums [100];
int invalid [100];
int num_invalid = 0;
for (int i=0;  i<number_of_customers;  i++)
    if  (cust_nums [i] < 1000  ||  cust_nums [i] > 9999)
    {
        num_invalid++;
        invalid [num_invalid] = cust_nums [i];
    }
```

Vector Code to Find Values

```
int cust_nums [100];
vector <int> invalid;
for (int i=0;  i<number_of_customers;  i++)
    if  (cust_nums [i] < 1000  ||  cust_nums [i] > 9999)
    {
        invalid.push_back (cust_nums [i]);
    }
```

Inserting and Deleting Elements from a Vector

- This is very similar to inserting and deleting elements from an array
- The difference is that now you can adjust the size of the vector using push_back() and pop_back()
- Very much like arrays, you can only insert and delete from the end of a vector
- So, the code remains essentially the same

Questions?

- Are there any questions?