CS 102
Introduction to Programming Using C++

Chapter 6

Arrays and Vectors

Homework

- Written homework
- R6.1, 2, 6, 10, 11, 13 14, 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

What Can You Do to an Array?
Example 1

- For this example, and several that follow suppose the array nums is declared

```
int nums[10];
```

- It is easy to initialize an array
- A for loop is a very convenient tool

```
for (int i=0;  i<10;  i++)
    nums [i] = 0;
```

OR

```
for (int i=0;  i<10;  i++)
    nums [i] = i;
```

Using Part of an Array

- The previous PowerPoint slideshow talked about using part of an array
- The rest of the slides do that
- Even though the array is dimensioned to 10, it may not be full
- We will use the int variable number_of_items to tell how many of the 10 items are in use

What Can You Do to an Array? Example 2

- You can
  - calculate the sum of the elements
  - calculate the average of the elements
  - find the highest element
  - find the lowest element
- as in the GetTemps-Array program

Example 2-Part 1
Calculating a Sum

- If you want to know the sum of the elements in an array, you use familiar code

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

Example 2-Part 2
Calculating an Average

- To calculate the average of the elements in an array, you just
  - calculate the sum of the elements
  - and divide by the number of elements

```
int sum = 0;
for (int i=0;  i<number_of_items;  i++)
    sum += nums[i];
double average = (double) sum / number_of_items;
```

Example 2-Part 3
Finding the Maximum and Minimum

- This code is familiar too

```
int max = nums [0];
int min = nums [0];
for (int i=1;  i<number_of_items;  i++)
{
    if  (nums[i] > max)
        max = nums [i];
    if  (nums[i] < min)
        min = nums [i];
}
```

Example 3
The Linear Search

- You can find something in the array
- A common technique for searching is the linear search
- You start by checking element 0
  - Then you check element 1
  - Then you check element 2, etc.
- You either find what you are looking for or you know it's not in the array

The Linear Search

- Again, this is familiar code
- We used the linear search with strings
- This is a slow search
  - However, it always works
  - And, it’s easy to code
- There are other ways to search, but they assume the data is in some order

The Linear Search-The Code

- You want to know if one of the integers in nums is 17
    - You also want to know where 17 is, if it's found
    - The value 17 is stored in the integer variable search_value

```
int pos = 0;
bool found = false;
while (pos < number_of_items  &&  !found)
   if (nums [pos] == search_value)
      found = true;
   else
      pos++;
```

Example 4
Copying an Array

- To copy the array nums to the array copy_of_nums, you cannot use

```
    copy_of_nums = nums;
```

- You must copy the array element by element

```
for (int i=0;  i<number_of_items;  i++)
   copy_of_nums [i] = nums[i];
```

Example 5
Listing the Elements in an Array

- To list the elements requires a separator
- Here is the code

```
cout << nums [0];
for (int i=1;  i<number_of_items;  i++)
   cout  <<  “,”  <<  nums[i];
```

OR

```
for (int i=0;  i<number_of_items-1;  i++)
    cout  <<  nums[i]  <<  “,”;
cout << nums [9];
```

Arrays As Data Structures

- Data structures are ways to store data
- There are two main types
  - An array is one of the types
- It is useful because it is easy to use
- Its main drawback is that it is not easily changed
  - The elements of the array can be changed
  - The array itself cannot be changed

Changing an Array Itself: Deleting a Position

- An array cannot be changed
- You cannot delete a position from an array
  - Suppose you have the array int scores[10];
  - You want to delete the third score
  - You want the array to have positions

    0, 1, 3, 4, 5, 6, 7, 8, 9, but not 2
  - You cannot do this

Changing an Array Itself: Inserting a Position

- Again, an array cannot be changed
- You also cannot insert a position into an array
  - Suppose you have the array int scores[10];
  - You decide to insert a score after the fourth score
  - You want the array to have positions

    0, 1, 2, 3, 3.5 maybe?, 4, 5, 6, 7, 8, 9
  - You cannot do this

Actually Changing an Array

- You can pretend to delete and insert
- Deleting and inserting can be very slow if you have a big array
- We will see code that deletes and inserts

Deleting an Element from an Array First Version

- Here is an easy and fast way to delete an element
- It assumes the order of the elements in the array is not important
  - A shopping cart is like this
  - Anything you put in a pile is like this
- If you don't care about the order, you could simply move the last element into the soon-to-be empty spot

```
scores [2] = scores [number_of_scores-1];
```

- Then you have to decrement number_of_scores

```
number_of_scores--;
```

Deleting an Element from an Array When Order Matters

- Again, the scenario is int scores[10];
  - Again, you want to delete the third score
- You can move the "bottom portion" of the array up to cover the third element
- In this case, we move

    scores [4] to scores [3]

    scores [5] to scores [4]

    etc. …

    scores [9] to scores [8]

Deleting an Element from an Array Second Version

- This code is for when order matters
- This code will be very slow if you have a big array
- Also, if you do this code often, it will slow down your program
    - For example, if it's in a loop
        - This means that there are two loops here!

```
for (int i=3;  i<number_of_scores;  i++)
    scores [i] = scores [i+1];
number_of_scores--;
```

Inserting an Element into an Array First Version

- Here is an easy and fast way to insert an element
- Again, we might assume the order of the elements in the array is not important
- If you don’t care about the order, you could simply insert the new item into the last spot

```
  scores [number_of_scores] = item;
```

- Then you have to increment number_of_scores

```
  number_of_scores++;
```

Inserting an Element Where It Belongs

- Again, the scenario is int scores[10];
- The elements of the array scores are sorted
  - For example, you might have entered your scores from smallest to largest
- You can move the "bottom portion" of the array down to open up a spot for the new element
- To insert a new element in position 2, we move

  scores [8] to scores [9]

  scores [7] to scores [8]

  etc. …

  scores [2] to scores [3]

Inserting an Element Where It Belongs Part 2

- Now, you can insert your item in position 2

  ```
  scores [2] = item;
  ```

- Then you have to increment number_of_scores

  ```
  number_of_scores++;
  ```

Inserting vs. Deleting

- When you delete, you have to move the array up
  - You start at the top and go down
- When you insert, you have to move the array down
  - You start at the bottom and go up
- It’s important to do this the correct way

Moving the Other Direction- Deleting

- What if you mix them up?
  - Let’s try to delete by moving the array up, but start at the bottom

    scores [9] goes to scores [8]

    scores [8] goes to scores [7]

    etc. …

    scores [3] goes to scores [2]
- What could go wrong?

Moving the Other Direction--Inserting

- Let’s also try to insert by moving the array down, but start at the top

    scores [2] goes to scores [3]

    scores [3] goes to scores [4]

    etc. …

    scores [8] goes to scores [9]

- What could go wrong?

Side Issues

- Both the insert and delete assume certain conditions
- The delete operation assumes the item you are trying to delete is actually in the array
  - Otherwise, you might try to delete from an empty array, or from a random position
- The insert operation assumes that there is room in the array to insert a new element
  - Otherwise, you have to “deny” the operation

Entering Data into an Array

- You can read numbers into an array using a loop
- If you know you will fill the array, you can use a for loop

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

Entering Data into an Array If You Don't Know How Many Input Items There Are

```
int number_of_scores = 0;
while (cin >> input)
{
    scores [number_of_scores] = input;
    number_of_scores++;
}
```

- The textbook also checks that there is room in the array before adding
- You should do that too!

Sorting

- Sorting is a painful operation because it takes a long time
- That’s why there are several ways to sort
- One easy sort to program is the selection sort
- it’s pretty easy to code

Swapping Two Data Items

- Suppose you want to swap the values of a and b
- Why doesn’t this code work?

```
a = b;
b = a;
```

- Instead, we do this

```
temp = a;      // Temporarily save a
a = b;         //  Copy b to a
b = temp;      //  Copy saved value of a to b
```

The Function index_of_smallest()

- To simplify the process, I will create and use a function
- I will call it index_of_smallest
  - It will be an integer function
  - It will have two integer arguments

  int index_of_smallest (posn1, posn2)

- It will find and return the index of the smallest element among

  scores [posn1, posn1+1, …, posn2]

A Function to Swap

- I will also create the function

  ```
  void swap (int &a, int &b)
  ```

- It will swap the values of its two arguments
- Notice the two ampersands

The Selection Sort

```
for (int i=0;  i<10;  ++i)
{
    //  Find the index of the smallest element in
    //  scores [i+1, i+2, …, 9]
    smallest_posn = index_of_smallest (i, 9);

    //  Swap element smallest_posn with the ith element
    swap (scores [i], scores [smallest_posn]);
}
```

An Example

- Let’s try this with a small array
- The declaration is int nums = {7, 5, 9, 2, 8};

- There are many sorting algorithms
- This is one of the slowest
- You can look up quicksort
- It’s one of the fastest sorting algorithms

The Binary Search

- Searching and sorting are two of the most time consuming operations a computer can do
- The linear search is the slowest search
- A much faster search is the binary search
  - It only works if the data is sorted
- The binary search works by breaking the data into two halves
- It finds which half the searched-for item appears in
- It discards the other half and repeats the process

Code for the Binary Search

- To perform the search, we need to keep

track of the left end and the right end

- In the example, we are searching through the array nums
- We are searching for search_value
- size is the number of elements in the array nums

The Code

```
bool found = false;
int low = 0;
int high = size - 1;
int pos = 0;
while (low <= high  &&  !found)
{
    pos = (low + high) / 2;
    if  (nums [pos] == search_value)
        found = true;
    else if  (nums [pos] < search_value)
        low = pos + 1;
    else
        high = pos - 1;
}
```

After the Loop

```
if  (found)
    cout << search_value << " found at position " << pos;
else
    cout << search_value
            << " not found.  Insert before position “ << pos;
```

Questions?

- Are there any questions?