Structured Programming
Using C++

Lecture 9: More on Arrays

Dr. Amal Khalifa

Lecture Contents:

- Arrays in Functions
  - Arrays as function arguments, return values (done!!)
  - Arrays in recursive functions

- Algorithms on Arrays
  - Searching
    - Linear Search
    - Binary Search
  - Sorting

- Multidimensional Arrays
Searching an Array

- Very typical use of arrays
  - Find the first occurrence of a given target value in a list

**Sample Dialogue**

Enter up to 20 nonnegative whole numbers.
Mark the end of the list with a negative number.
10 20 30 40 50 60 70 80 -1
Enter a number to search for: 10
10 is stored in array position 0
(Remember: The first position is 0.)
Search again? (y/n followed by Return): y
Enter a number to search for: 40
40 is stored in array position 3
(Remember: The first position is 0.)
Search again? (y/n followed by Return): y
Enter a number to search for: 42
42 is not on the list.
Search again? (y/n followed by Return): n
End of program.

---

Searching an Array (1 of 3)

**Display 5.6  Searching an Array**

```cpp
1  //Searches a partially filled array of nonnegative integers.
2  #include <iostream>
3  using namespace std;
4  const int DECLARED_SIZE = 20;
5  void fillArray(int a[], int size, int& numberUsed);
6  //Precondition: size is the declared size of the array a.
7  //Postcondition: numberUsed is the number of values stored in a.
8  //a[0] through a[numberUsed-1] have been filled with
9  //nonnegative integers read from the keyboard.
10 int search(const int a[], int numberUsed, int target);
11  //Precondition: numberUsed is <= the declared size of a.
12  //Also, a[0] through a[numberUsed -1] have values.
13  //Returns the first index such that a[index] == target,
14  //provided there is such an index; otherwise, returns -1.
```

---

Dr. Amal Khalifa - Spring 2012
Searching an Array (2 of 3)

```cpp
int main( )
{
    int arr[DECLARED_SIZE], listSize, target;
    fillArray(arr, DECLARED_SIZE, listSize);
    char ans;
    int result;
    do
    {
        cout << "Enter a number to search for: ";
        cin >> target;
        result = search(arr, listSize, target);
        if (result == -1)
            cout << target << " is not on the list.
        else
            cout << target << " is stored in array position 
        << result << endl
            << "(Remember: The first position is 0.
    
```

Searching an Array (3 of 3)

```cpp
void fillArray(int a[], int size, int& numberUsed)
{ The rest of the definition of fillArray is given in Display 5.5
}
int search(const int a[], int numberUsed, int target)
[
    int index = 0;
    bool found = false;
    while (((!found) && (index < numberUsed)))
    {
        if (target == a[index])
            found = true;
        else
            index++;
    if (found)
        return index;
    else
        return -1;
}
Recursive Search

- Recursive function to search array
  - Determines if item is in list
  - if found:
    - Where in list it is
- Assumes array is sorted
- Breaks list in half
  - Determines if item in 1st or 2nd half
  - Then searches again just that half
    - Recursively (of course)!

Pseudo code for Binary Search

```c
int a[Some_Size_Value];

ALGORITHM TO SEARCH a[first] THROUGH a[last]
//Precondition:
//a[first] <= a[first + 1] <= a[first + 2] <= ... <= a[last]

TO LOCATE THE VALUE key:
if (first > last) //A stopping case
  found = false;
else
  {  
    mid = approximate midpoint between first and last;
    if (key == a[mid]) //A stopping case
    {  
      found = true;
      location = mid;
    }
    else if key < a[mid] //A case with recursion
    search a[first] through a[mid - 1];
    else if key > a[mid] //A case with recursion
    search a[mid + 1] through a[last];
  }
```

Dr. Amal Khalifa, 2012
Tracing !!

- Notice binary search algorithm actually solves "more general" problem
  - Original goal: design function to search an entire array
  - Our function: allows search of any interval of array
    - By specifying bounds first and last

- Very common when designing recursive functions

---

Checking the Recursion

- Check binary search against criteria:
  1. No infinite recursion:
     - Each call increases first or decreases last
     - Eventually first will be greater than last
  2. Stopping cases perform correct action:
     - If first > last → no elements between them, so key can't be there!
     - IF key == a[mid] → correctly found!
  3. Recursive calls perform correct action
     - If key < a[mid] → key in 1st half – correct call
     - If key > a[mid] → key in 2nd half – correct call
Efficiency of Binary Search

- Extremely fast
  - Compared with sequential search
- Half of array eliminated at start!
  - Then a quarter, then 1/8, etc.
  - Essentially eliminate half with each call
- Example:
  Array of 100 elements:
  - Binary search never needs more than 7 compares!
    - Logarithmic efficiency (log n)

Sorting an Array (optional)

Display 5.7  Selection Sort

```
| 8 | 6 | 10 | 2 | 16 | 4 | 18 | 14 | 12 | 20 |
```

```
| 8 | 6 | 10 | 2 | 16 | 4 | 18 | 14 | 12 | 20 |
```

```
| 2 | 6 | 10 | 8 | 16 | 4 | 18 | 14 | 12 | 20 |
```

```
| 2 | 6 | 10 | 8 | 16 | 4 | 18 | 14 | 12 | 20 |
```

```
| 2 | 4 | 10 | 8 | 16 | 6 | 18 | 14 | 12 | 20 |
```
Multidimensional Arrays

- Array of arrays
- Arrays with Two indexes
- Example:
  ```
  int balances[11][6];
  ```
- C++ allows any number of indexes
  - Example?
  - Index??
  - Typically no more than two

Multidimensional Array Parameters

- Similar to one-dimensional array
  - 1st dimension size not given
    - Provided as second parameter
  - 2nd dimension size IS given

```c
void DisplayPage(const char p[][100], int sizeD1)
{
    for (int index1=0; index1<sizeD1; index1++)
    {
        for (int index2=0; index2 < 100; index2++)
            cout << p[index1][index2];
        cout << endl;
    }
}
```
Example: TWO-DIMENSIONAL GRADING PROGRAM

Write a program that uses a two-dimensional array named grade to store the grade records for a small class. (The first array index is used to designate a student, and the second array index is used to designate a Quiz).

The program should be able to:
- Store the grade data for ten students and three quizzes
- Compute the average quiz score for each of the students
- Compute the average score on each quiz.
- Display the grade records for the entire class

The grades Array

- Two indices:
  - the first used to designate a student,
  - the second is used to designate a quiz.
  - numbered starting with 0 rather than 1,
  - must subtract 1 from the number to obtain the index

Example: the score that student number 4 received on quiz 1 is recorded in grade[3][0].
Average Arrays

- **Array** \( \text{stAve} \):
  - used to record the average quiz score for each of the students.
  - Example, \( \text{stAve}[1] \) equal to the average of the quiz scores received by student 2,

- **Array** \( \text{quizAve} \):
  - used to record the average score for each quiz.
  - Example: \( \text{quizAve}[0] \) equal to the average of all the student scores for quiz 1.

That’s all for today!!

Thanks.....