Arrays 2

CS 16: Solving Problems with Computers I
Lecture #12

Ziad Matni
Dept. of Computer Science, UCSB

MIDTERM #2 IS COMING! November 14th!

- Material: Post-Midterm #1 *Lecture 7 thru 12*
  - Homework, Labs, Lectures, Textbook
- **Tuesday, 11/14** in this classroom
- **Starts at 2:00pm **SHARP** (come early)**
- **Ends at 3:15pm **SHARP****
- **BRING YOUR STUDENT IDs WITH YOU!!!**
- Closed book: no calculators, no phones, no computers
- Only 1 sheet (single-sided) of written notes
  - Must be no bigger than 8.5" x 11"
  - **You have to turn it in with the exam**
- **You will write your answers on the exam sheet itself.**

What's on Midterm #2?

From the book:
Chapter 5.4, 5.5, 6, 7, and 8.1, 8.2

- Test and Debugging Functions
- General Debug Techniques
- Numerical (all combos of bin, oct, hex, and dec) Conversions
- C-Strings
- Character Manipulators
- C++ Strings

  … and **all** its built-in member functions that we discussed
- File I/O

  … and **all** the many built-in member functions & manipulators that we discussed, both character-based and string/line-based

  … don't forget all the different ways to read an input file until it ends, or how to check to see if an I/O file is valid or not
- Arrays

Lecture Outline

- Arrays and Functions
  - Using the **const** Modifier in a Function when passing arrays
  - Returning arrays in a Function

- Multidimensional Arrays
- Partially Filled Arrays
- Searching Arrays

On the midterm

Not on the midterm

- Compiling Multiple Files and Using **Makefile**

Changing The Values In An Array

- Array parameters allow a function to **change the values** stored in the array argument

- Similar to how a parameter being passed by reference would be
- This is because an array is structured by C++ *as a pointer* (more on this later)

- **Example:**

```
void show_the_world(int a[ ], int size);
// actually changes the array
```

const Modifier

- If you want a function to ***not change*** the values of the array argument, use the modifier **const**

- An array parameter modified with **const** is called a *constant array parameter*

- Example:

```
void show_the_world(const int a[ ], int size);
```

Using **const** With Arrays

- If **const** is used to modify an array parameter:
  - **const** has to be used in *both* the function *declaration* and *definition*

- The compiler will issue an error if you write code that changes the values stored in the array parameter
  - In other words, don't even try to re-write any part of the array if you've passed it into a function using **const**

Returning An Array

- Recall that functions can return a value of type int, double, char, …, or even a class type (like string)
- **BUT functions cannot return arrays**
- We'll learn later how to return a ***pointer*** to an array instead…

Summary Difference

```
void thisFunction(int arr[ ], int size);
```

Array "arr" gets passed and whatever changes are done inside the function will result in changes to "arr" where it's called.

```
void thisFunction(const int arr[ ], int size);
```

Array "arr" gets passed BUT whatever changes are done inside the function will NOT result in changes to "arr" where it's called.

```
int* thisFunction(int arr[ ], int size);
```

Array "arr" gets passed and whatever changes are done inside the function will result in changes to "arr" where it's called. ADDITIONALLY, a new *pointer* to an array "thisFunction" is passed back (DON'T WORRY ABOUT THIS UNTIL AFTER WE LEARN ABOUT POINTERS!) ***It's not on the midterm…***

Multi-Dimensional Arrays

- C++ allows arrays with multiple index dimensions
- EXAMPLE: `char page[30][100];`
  declares an array of characters named **page**
  - **page** has two index values:
    The 1st ranges from 0 to 29
    The 2nd ranges from 0 to 99
  - Each index in enclosed in its own brackets

[0][0]	[0][1]	...	[0][98]	[0][99]
[1][0]	[1][1]	...	[1][98]	[1][99]
...	...	...	...	...
[28][0]	[28][1]	...	[28][98]	[28][99]
[29][0]	[29][1]	...	[29][98]	[29][99]

- Page can be visualized as an array of 30 rows and 100 columns
  - **page** is actually an array of size 30
  - **page's base type** is an array of 100 characters

Program Example: Grading Program

- Grade records for a class can be stored in a two-dimensional array
- A class with 4 students and 3 quizzes the array could be declared as

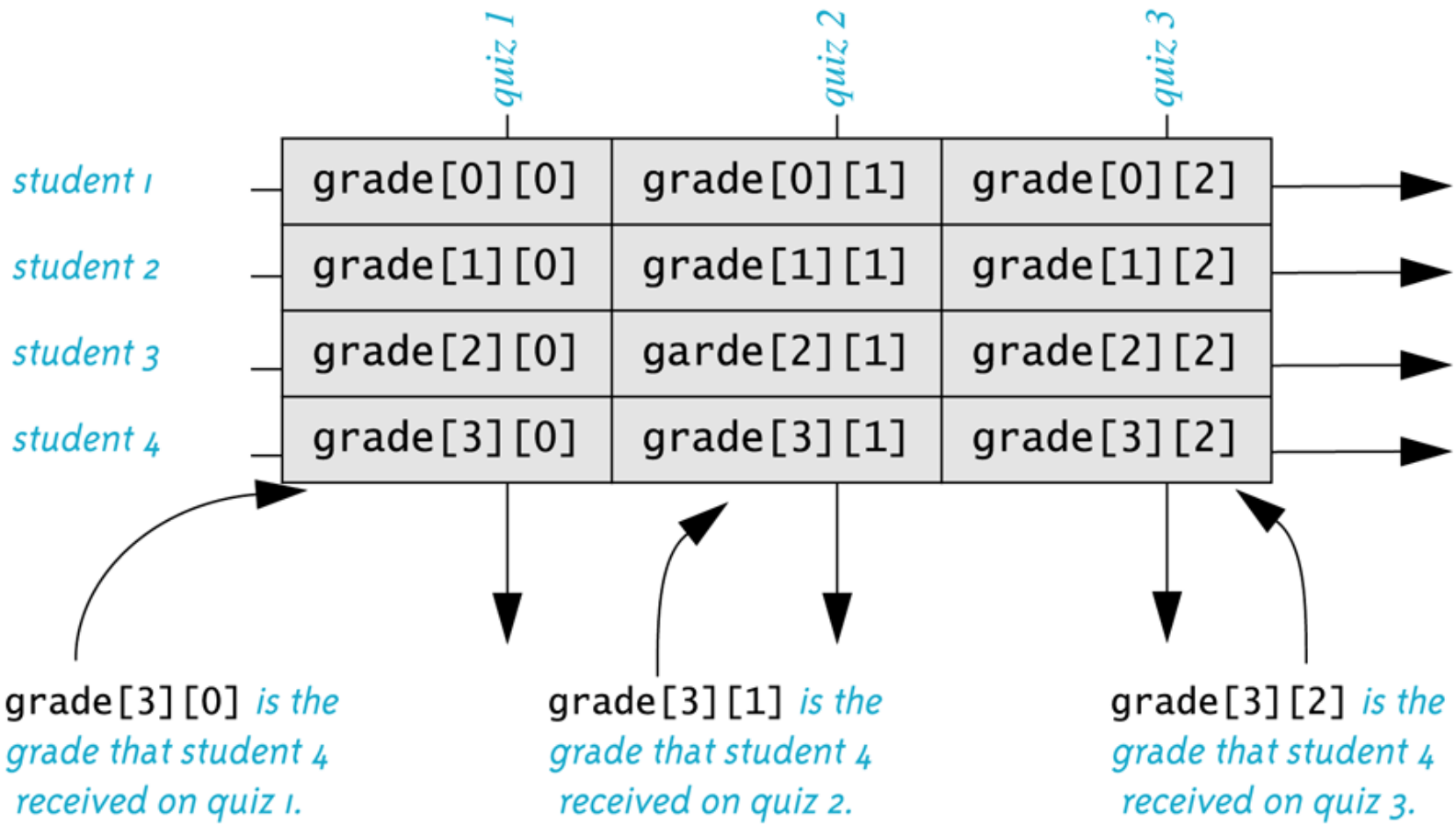
```
int grade[4][3];
```

Each student (0 thru 3) has 3 grades (0 thru 2)

  - The first array index refers to the number of a student
  - The second array index refers to a quiz number

- Your textbook, Ch. 7, Display 7.14 has an example

The Two-Dimensional Array grade



Use Nested **for-loops** to Go Through a MDA

Example:

```
const int MAX1 = 10, MAX2 = 20;
int arr[MAX1][MAX2];
…
for (int i = 0; i < MAX1; i++)
   for (int j = 0; j < MAX2; j++)
      cout << arr[i][j];
```

Initializing MDAs

- Recall that you can do this for uni-dimensional arrays and get all elements initialized to zero:

```
double numbers[100] = {0};
```

- For multidimensional arrays, it's similar syntax:

```
double numbers[5][100] = { {0}, {0} };
double numbers[5][100] = {0}; // This ALSO works!
```

- What would this do?

```
double numbers[2][3] = { {6,7}, {8,9} };
```

Multidimensional Array Parameters in Functions

- Recall that the size of an array is not needed when declaring a formal parameter:

```
void display_line(char a[ ], int size);
```

Look! No size!

Size is here instead!

- BUT the **base type must be completely specified in the parameter declaration** of a multi-dimensional array

```
void display_page(char page[ ] [100], int size_dimension1);
```

Base has a size def.!

Programming With Arrays

- The size requirement for an array might need to be **un-fixed**
  - Often varies from one run of a program to another
  - Size is often *not known* when the program is written

- A common solution to the size problem (while still using "regular" arrays):
  - Declare the array size to be the **largest** that could be needed
  - Decide how to deal with *partially filled arrays*
    - Example forthcoming…

Partially Filled Arrays

- When using arrays that are partially filled…
  - Functions dealing with the array may not need to know the **declared size of the array**
  - Only **how many maximum number of elements** need to be stored in the array!

- A parameter - let's call it **number_used** - may be sufficient to ensure that referenced index values are legal

Partially Filled Array (*part 3 of 3*)

Sample Dialogue

```
This program reads golf scores and shows
how much each differs from the average.
Enter golf scores:
Enter up to 10 nonnegative whole numbers.
Mark the end of the list with a negative number.
69 74 68 -1
Average of the 3 scores = 70.3333
The scores are:
69 differs from average by -1.33333
74 differs from average by 3.66667
68 differs from average by -2.33333
```

```
#include <iostream>
const int MAX_NUMBER_SCORES = 10;

void fill_array(int a[], int size, int& number_used);

double compute_average(const int a[], int number_used);
void show_difference(const int a[], int number_used);
int main()
{
    using namespace std;
    int score[MAX_NUMBER_SCORES], number_used;

    cout << "This program reads golf scores and shows\n"
         << "how much each differs from the average.\n";

    cout << "Enter golf scores:\n";
    fill_array(score, MAX_NUMBER_SCORES, number_used);
    show_difference(score, number_used);

    return 0;
}

//Uses iostream:
void fill_array(int a[], int size, int& number_used)
{
    using namespace std;
    cout << "Enter up to " << size << " nonnegative whole numbers.\n"
         << "Mark the end of the list with a negative number.\n";

    int next, index = 0;
    cin >> next;
    while ((next >= 0) && (index < size))
    {
        a[index] = next;
        index++;
        cin >> next;
    }

    number_used = index;
}
```

```
double compute_average(const int a[], int number_used)
{
    double total = 0;
    for (int index = 0; index < number_used; index++)
        total = total + a[index];
    if (number_used > 0)
    {
        return (total/number_used);
    }
    else
    {
        using namespace std;
        cout << "ERROR: number of elements is 0 in compute_average.\n"
             << "compute_average returns 0.\n";
        return 0;
    }
}

void show_difference(const int a[], int number_used)
{
    using namespace std;
    double average = compute_average(a, number_used);
    cout << "Average of the " << number_used
         << " scores = " << average << endl
         << "The scores are:\n";
    for (int index = 0; index < number_used; index++)
    cout << a[index] << " differs from average by "
         << (a[index] - average) << endl;
}
```

Your textbook, Ch. 7
Display 7.9

Searching Arrays

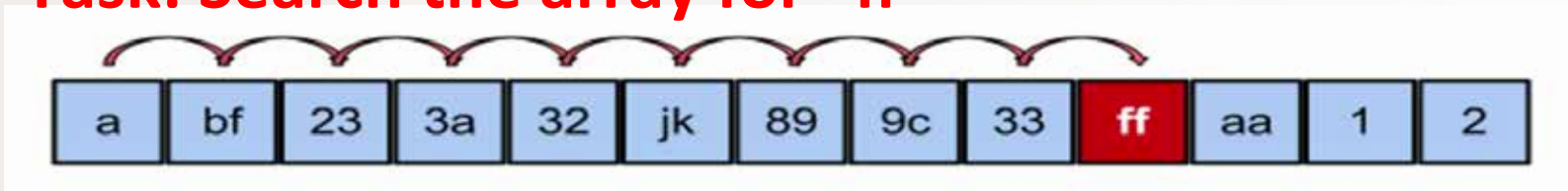
- A **sequential search** is one way to search an array for a given value. The algorithm is as follows:

1. Look at each element from first to last to see if the target value is equal to any of the array elements
2. The index of the target value is returned to indicate where the value was found in the array
3. A value of -1 is returned if the value was not found anywhere

Pros? Cons?

Sequential Search

Task: Search the array for "ff"



ARRAY a[]: a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11] a[12]

```
int SeqSearch
(int arr[], int array_size, int target)
{
   int index(0);
   bool found(false);
   while ((!found) && (index < array_size))
   {
      if (arr[index] == target)
         found = true;
      else
         index++;
   }
   if (found)
      return index;
   else
      return -1;
}
```

Simple Sequential Search Function Example

Searching an Array ***(part 1 of 2)***

```
//Searches a partially filled array of nonnegative integers.
#include <iostream>
const int DECLARED_SIZE = 20;

void fill_array(int a[], int size, int& number_used);
//Precondition: size is the declared size of the array a.
//Postcondition: number_used is the number of values stored in a.
//a[0] through a[number_used-1] have been filled with
//nonnegative integers read from the keyboard.

int search(const int a[], int number_used, int target);
//Precondition: number_used is <= the declared size of a.
//Also, a[0] through a[number_used -1] have values.
//Returns the first index such that a[index] == target,
//provided there is such an index; otherwise, returns -1.

int main( )
{
    using namespace std;
    int arr[DECLARED_SIZE], list_size, target;

    fill_array(arr, DECLARED_SIZE, list_size);

    char ans;
    int result;
    do
    {
        cout << "Enter a number to search for: ";
        cin >> target;

        result = search(arr, list_size, target);
        if (result == -1)
            cout << target << " is not on the list.\n";
        else
            cout << target << " is stored in array position "
                 << result << endl
                 << "(Remember: The first position is 0.)\n";

        cout << "Search again?(y/n followed by Return): ";
        cin >> ans;
    }while ((ans != 'n') && (ans != 'N'));

    cout << "End of program.\n";
    return 0;
}
```

Searching an Array ***(part 2 of 2)***

```
//Uses iostream:
void fill_array(int a[], int size, int& number_used)
<The rest of the definition of fill_array is given in Display 10.9.>

int search(const int a[], int number_used, int target)
{

    int index = 0;
    bool found = false;
    while ((!found) && (index < number_used))
        if (target == a[index])
            found = true;
        else
            index++;

    if (found)
        return index;
    else
        return -1;
}
```

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.
```

Midterm #2 Sample Question 1

Given the code below, what will the output be if the user enters:

Hola Amigo!!

```
string MyString;
cin >> MyString;
for (int j = 0; j < 3; j++)
   MyString[j] = toupper(MyString[j+1])
cout << MyString << endl;
```

ANS: OLAa

Midterm #2 Sample Question 2

Given the code below, what will the output be if the user enters:

My phone number's 805-555-1212. Call me?

```
string TestString, NewTestString="";
getline(cin, TestString);
for (int i = 0; i < TestString.size(); i++)
   if (isalpha(TestString[i]) || isspace(TestString[i]))
      NewTestString += TestString[i];
cout << NewTestString << endl;
```

ANS:
My phone numbers Call me

2 spaces

Midterm #2 Sample Question 3

Describe what this function returns or intends to return?
If there's an error, point out if it's a **compilation** error or a **run-time** error or a **design** mistake.

```
double ArrayCalc(int arr[], int arrSize)
{
   double sum; // why "double"???
   for (int j = 1; j < arrSize; j++)
      sum += arr[j];
   return (sum/arrSize);
}
```

ANS:
It looks like it is intended to return the average of all values of an integer array. However, it has 2 design mistake:
(1) It begins with array index 1, not 0.
(2) It does not initialize sum to 0 before it starts accumulating numbers in it.

Midterm #2 Sample Question 4

What will this code output? If there's an error, point out if it's a **compilation** error or a **run-time** error or a **design** mistake.

```
double Balances[10] = {0};
for (int n = 0; n < 10; n += 2)
   Balances[n] = 5;
for (int n = 0; n < 10; n++)
   cout << Balances[n];
```

ANS: 5050505050 (no error)

YOUR TO-DOs

- ❑ **STUDY FOR YOUR MIDTERM #2 EXAM!!!! (on Tue. 11/14)**
- ❑ HW 7 is now out and due Thu. 11/16
- ❑ Lab 6 due Fri. 11/10

- ❑ Visit Prof's and TAs' office hours if you need help!

</LECTURE>