# More on Arrays

**CS 16: Solving Problems with Computers I**
**Lecture #13**

Ziad Matni
Dept. of Computer Science, UCSB

# Announcements

- **Homework #12 due today**
- **No homework assigned today!! ☺**
- **Lab #7 is due on Monday, 11/14 at 8 AM**

- **Midterm #2 is this Thursday in class**

# Midterm 2

**Material to Review:**

- Lectures: Start at lecture #7, end at lecture #13
- Book Sections: Chapters 5, 6, 7, 8.1, 8.2, 8.3 (maybe?)
- Homework: Start at HW6, end at HW13
- Labs: Lab4, Lab5, Lab6

**Covers**:

- Functions
- I/O Streams (including command line inputs)
- Binary, Decimal, Octal, Hexadecimal Conversions
- Characters and Strings
- Arrays
- Vectors (if we get to it in time)

# MIDTERM 2 IS COMING!

- **Thursday, 11/10** in this classroom
- **Starts at 2:00pm **SHARP****
- **I will chose where you sit!**
- Duration: **1 hour long**


- 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.**

# Lecture Outline

- Programming with Arrays
- Multidimensional Arrays
- Lab 7 Questions

# Programming With Arrays

- The size needed for an array is changeable
  - 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:
  - 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 elements** are stored in the array
  - A parameter, **number_used**, may be sufficient to ensure that referenced index values are legal

```
#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**

## 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
```

# Constants as Arguments

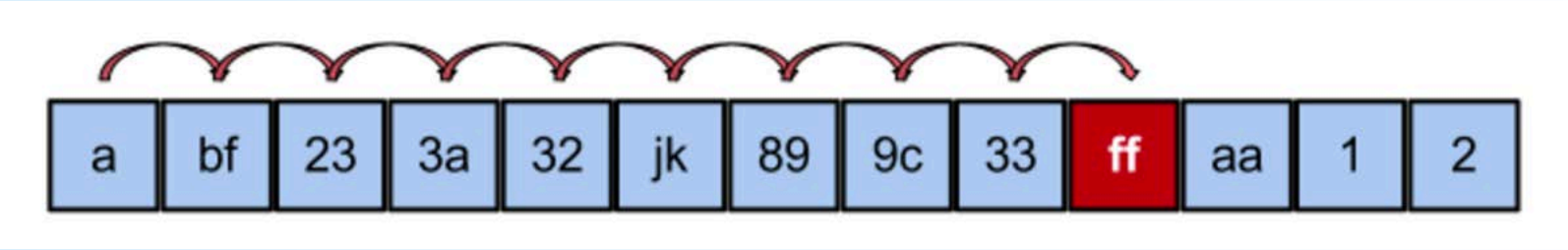
- When function fill_array (Display 7.9) is called, MAX_NUMBER_SCORES is used as an argument
  - Can't MAX_NUMBER_SCORES be used directly without making it an argument?
    - Using MAX_NUMBER_SCORES as an argument makes it clear that fill_array requires the array's declared size
    - This makes fill_array easier to be used in other programs

# Searching Arrays

- A sequential search is one way to search an array for a given value
  - Look at each element from first to last to see if the target value is equal to any of the array elements
  - The index of the target value can be returned to indicate where the value was found in the array
  - A value of -1 can be returned if the value was not found

# 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]

# Example **search** Function

*(See Display 7.10 in the textbook)*

- Uses a *while* loop to compare array elements to the target value
- Sets a variable of type **bool** to true if the target value is found, ending the loop
- Checks the **bool** variable when the loop ends to see if the target value was found
- Returns the index of the target value if found, otherwise returns -1

## 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.
```


DEMO

# Program Example: Sorting an Array

- Sorting a list of values is very common task
  - Create an alphabetical listing
  - Create a list of values in ascending order
  - Create a list of values in descending order
- Many sorting algorithms exist
  - Some are very efficient
  - Some are easier to understand

# Program Example: The Selection Sort Algorithm

- When the sort is complete, the elements of the array are ordered such that

  **a[0] < a[1] < … < a [ number_used -1]**

- This leads to an outline of an algorithm:

  for (int index = 0; **index < number_used**; index++)

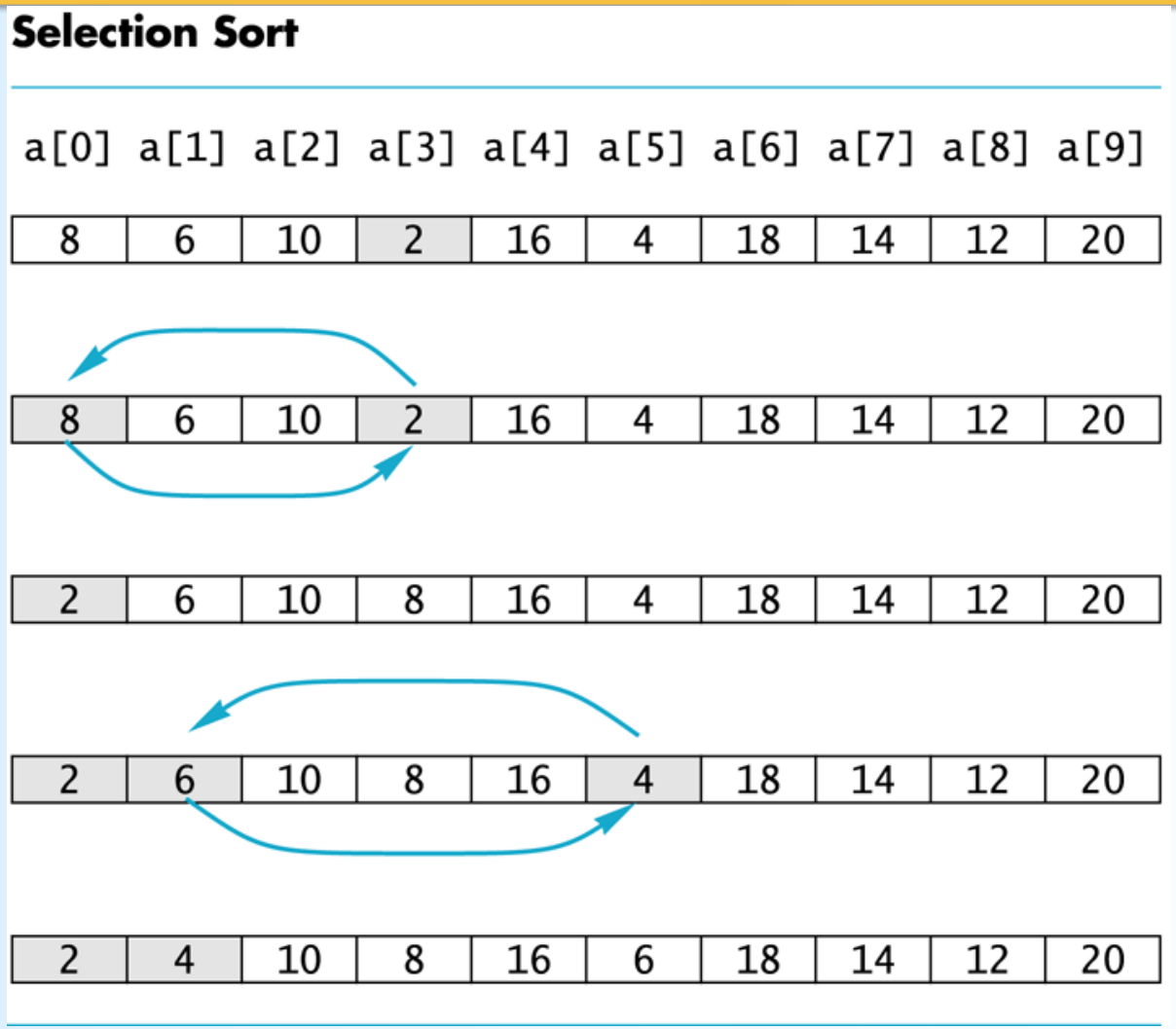
  place the index$^{th}$ smallest element in a[index]

# Program Example: Sort Algorithm Development

*(See Display 7.10 in the textbook)*

- One array is sufficient to do our sorting
- Search for the *smallest* value in the array
- Place this value in a[0], and place the value that was in a[0] in the location where the smallest was found
  - i.e. swap them
- Starting at a[1], find the smallest remaining value swap it with the value currently in a[1]
- Starting at a[2], continue the process until the array is sorted

# Sort from smallest to largest



## DISPLAY 7.12 Sorting an Array *(part 1 of 2)*

```
1  //Tests the procedure sort.
2  #include <iostream>

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

8  void sort(int a[], int number_used);
9  //Precondition: number_used <= declared size of the array a.
10 //The array elements a[0] through a[number_used - 1] have values.
11 //Postcondition: The values of a[0] through a[number_used - 1] have
12 //been rearranged so that a[0] <= a[1] <= ... <= a[number_used - 1].

13 void swap_values(int& v1, int& v2);
14 //Interchanges the values of v1 and v2.

15 int index_of_smallest(const int a[], int start_index, int number_used);
16 //Precondition: 0 <= start_index < number_used. Referenced array elemen
17 //values.
18 //Returns the index i such that a[i] is the smallest of the values
19 //a[start_index], a[start_index + 1], ..., a[number_used - 1].

20 int main( )
21 {
22     using namespace std;
23     cout << "This program sorts numbers from lowest to highest.\n";

24     int sample_array[10], number_used;
25     fill_array(sample_array, 10, number_used);
26     sort(sample_array, number_used);

27     cout << "In sorted order the numbers are:\n";
28     for (int index = 0; index < number_used; index++)
29         cout << sample_array[index] << " ";
30     cout << endl;

31     return 0;
32 }

33 //Uses iostream:
34 void fill_array(int a[], int size, int& number_used)

35 void sort(int a[], int number_used)
36 {
37     int index_of_next_smallest;
```

<The rest of the definition of `fill_array` is given in Display 7.9.>

*(co*

## DISPLAY 7.12 Sorting an Array *(part 2 of 2)*

```
38     for (int index = 0; index < number_used - 1; index++)
39     {//Place the correct value in a[index]:
40         index_of_next_smallest =
41                         index_of_smallest(a, index, number_used);
42         swap_values(a[index], a[index_of_next_smallest]);
43         //a[0] <= a[1] <=...<= a[index] are the smallest of the original array
44         //elements. The rest of the elements are in the remaining positions.
45     }
46 }
47

48 void swap_values(int& v1, int& v2)
49 {
50     int temp;
51     temp = v1;
52     v1 = v2;
53     v2 = temp;
54 }
55

56 int index_of_smallest(const int a[], int start_index, int number_used)
57 {
58     int min = a[start_index],
59         index_of_min = start_index;
60     for (int index = start_index + 1; index < number_used; index++)
61         if (a[index] < min)
62         {
63             min = a[index];
64             index_of_min = index;
65             //min is the smallest of a[start_index] through a[index]
66         }
67

68     return index_of_min;
69 }
```

### *Sample Dialogue*

```
This program sorts numbers from lowest to highest.
Enter up to 10 nonnegative whole numbers.
Mark the end of the list with a negative number.
80 30 50 70 60 90 20 30 40 -1
In sorted order the numbers are:
20 30 30 40 50 60 70 80 90
```

# Multi-Dimensional Arrays

- C++ allows arrays with multiple index values
  - char **page** [30] [100];
    declares an array of characters named **page**
    - page has two index values:
      The first ranges from 0 to 29
      The second ranges from 0 to 99
  - Each index  in enclosed in its own brackets
  - Page can be visualized as an array of
    30 rows and 100 columns

# Index Values of page

- The indexed variables for array page are
  page[0][0], page[0][1], …, page[0][99]
  page[1][0], page[1][1], …, page[1][99]
- …
  page[29][0], page[29][1], … , page[29][99]

- page is actually an array of size 30
  - page's base type is an array of 100 characters

# Multidimensional Array Parameters

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

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

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

```
void display_page(char page[ ] [100],
                                  int size_dimension_1);
```

# Program Example: Grading Program

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

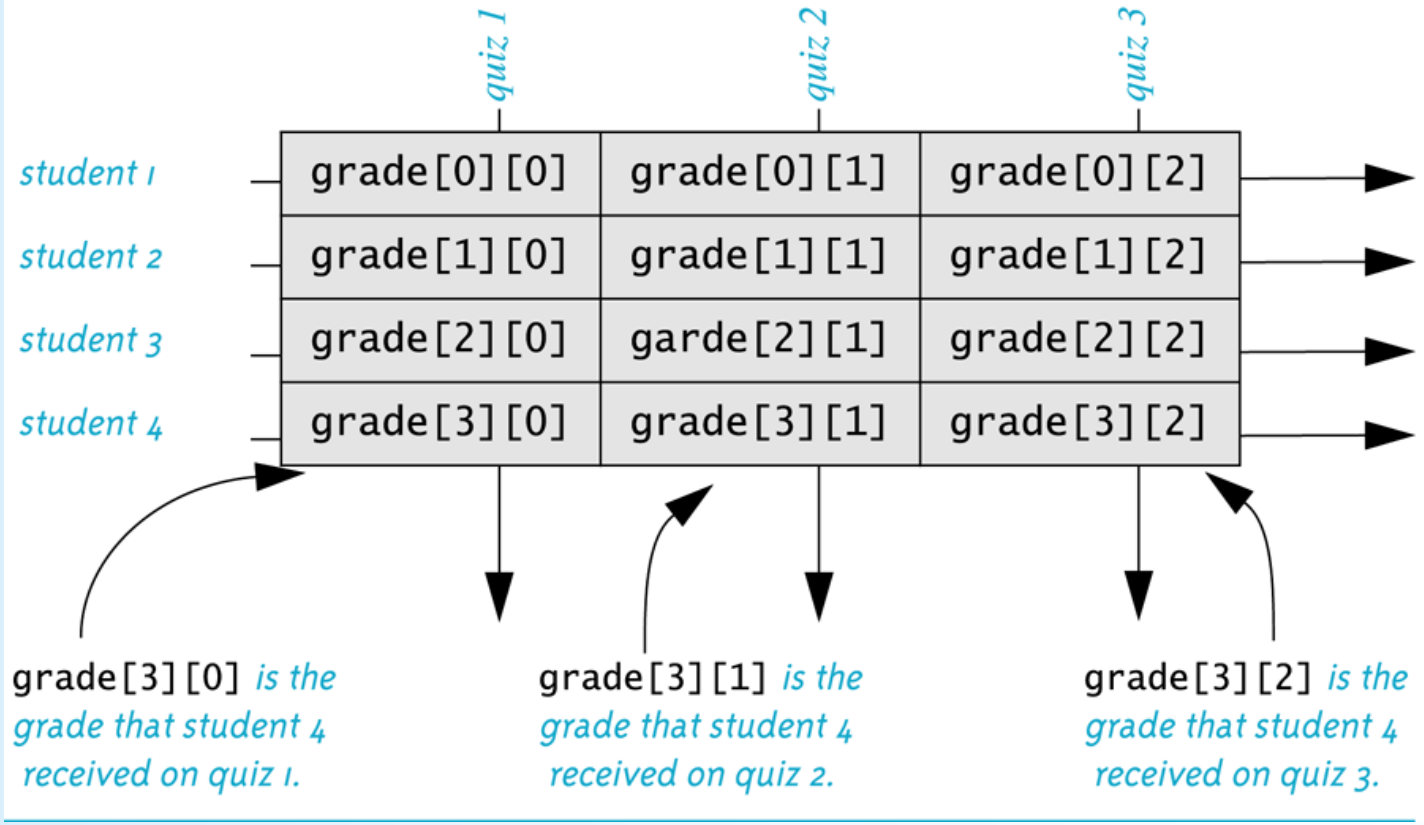
```
int grade[4][3];
```

Each student (1 thru 4) has 3 grades (1 thru 3)

    - The first array index refers to the number of a student
    - The second array index refers to a quiz number
- Since student and quiz numbers start with one, we subtract one to obtain the correct index

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

## The Two-Dimensional Array grade



# </LECTURE>

# Lab 7

- Partner-up (optional)
- Both exercises deal with 2-D arrays