

# **Review of How to Solve a Problem in C++**

## **Arrays**

**CS 16: Solving Problems with Computers I**  
**Lecture #12**

Ziad Matni  
Dept. of Computer Science, UCSB

# Announcements

- **Homework #11 due today**
- **Lab #6 is due on Friday at Noon**
  - Lab #7 will be due on Monday, 11/14 at 8 AM
- **Grades:**
  - Homework **1 thru 7** are up
  - Midterm **1** will be up by end of today
    - To review exam, please go to the designated TA's office hours (see next page)
- Midterm 2 is in 1 week: **Thursday, 11/10**

# Midterm 1

- Average = 90
  - Niiiiiiiice
- When grades will be posted, I will announce them via mass email to class (thru Gauchospace)
- You'll see a letter next to your grade under "Version":  
A, B, C, D
  - This isn't your letter grade! 😊
  - This will tell you which TA has your exam
- You may view and discuss exams with your TAs or myself, but **you may not take the exams anywhere.**

# 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)

# Lecture Outline

---

- Review of Lab 6 Problems
  - Focusing on one of them
- Introduction to Arrays (Ch. 7)
- Arrays in Functions
- Algorithmic Designs with Arrays

# Lab 6

- `stddev.cpp` 20 pts
  - Get numbers (double) inputs from a file
  - Calculate:  $s = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$
- `operators.cpp` 40 pts
  - Read a file with “bad” cin and cout uses
  - Write a file with all lines from original, only with fixes
- `binconverter.cpp` 40 pts

# binconverter

---

## Requirements:

- Get inputs from a file with binary numbers
- Output conversion to decimal, hexadecimal, and octal numbers.
- Let's come up with a plan...

# High Level View

---

**DON'T THINK IN C++ JUST YET!  
THIS IS A PROBLEM TO SOLVE!  
SO BREAK IT DOWN! DIVIDE AND CONQUER!**

1. Get binary number from file
2. Convert to decimal
3. Convert to octal
4. Convert to hexadecimal
5. Show outputs
6. Profit?!



# 1. Get binary number

1. Get binary number from file
2. Convert to decimal
3. Convert to octal
4. Convert to hexadecimal
5. Show outputs

- A. Define *ifstream* variable
- B. Get filename from user
- C. Open input file
  - a) Check to see that no errors occur
- D. Read in the binary number as a string
- E. Send on to Part 2

## 2. Convert to decimal

1. Get binary number from file
2. Convert to decimal
3. Convert to octal
4. Convert to hexadecimal
5. Show outputs

A. How do I convert bin  $\rightarrow$  dec?

a) Position Notation!

$$b_0x2^0 + b_1x2^1 + b_2x2^2 + \dots + b_nx2^n$$

*(it's a sum of products)*

*Example:*

1	1	0	1	0

B. Read in one bit (i.e. character) at a time

C. Multiply by the appropriate  $2^n$  value

a)  $1 \times 2$ ,  $1 \times 8$ ,  $1 \times 16$

D. Add up all the products

a) I get 25

# 3. Convert to octal

1. Get binary number from file
2. Convert to decimal
3. Convert to octal
4. Convert to hexadecimal
5. Show outputs

A. How do I convert bin  $\rightarrow$  octal?

- a) Collect 3 bits at a time!
- b) Remember: start from the LSB

*Example:*

1	1	0	1	0
---	---	---	---	---

B. In this example, collection is:

1 1 & 0 1 0

C. Convert those to a decimal number

- a) We did this function already!!!
- b) Recall: All symbols in octal exist in decimal as well

D. Put the new converted numbers side by side

- a) I get 32.

# 4. Convert to hex

1. Get binary number from file
2. Convert to decimal
3. Convert to octal
4. Convert to hexadecimal
5. Show outputs

A. How do I convert bin  $\rightarrow$  hex?

a) Collect 4 bits at a time!

B. In this example, collection is:

*Example:*

1	1	0	1	0	
1	&	1	0	1	0

C. Convert those to a decimal number

a) We did this function already!!!

b) Recall: Symbols in hex encompass decimal symbol + 6 others

D. Put the new converted numbers side by side

a) I get 1 and 10, but I know that 10 in hex is A, so...

b) I actually get 1A.

# 5. Show outputs

1. Get binary number from file
2. Convert to decimal
3. Convert to octal
4. Convert to hexadecimal
5. Show outputs

A. Requirement is to output to screen

a) Easy-peasy

# NOW... Think in C++

- Look at your plan/pseudocode

```
string.length()    returns the length of a string, i
                  so if string = "code", string.length() = 4.
string[n]          returns the nth character in the string (indexing starts at 0),
                  so string[0] = "c", string[1] = "o", etc...
int(char c)        converts a character into its ASCII code,
                  so if c = 'a', then int(c) = 97,
                  and if c = 'c', the int(c) = 99, etc...
to_string(int i)   converts an integer into a string. Function found in <string> library.
                  Example, if i = 73, then to_string(i) = "73"
```

- **For each entry think of**
  - How you'll code it
  - How you'll put in the requirements
  - How you'll test it

# How/When to Use the Required Functions

## string.length()

- Useful in determining how many loops a for/while statement should have

```
string.length() returns the length of a string, i
                so if string = "code", string.length() = 4.
string[n]       returns the nth character in the string (indexing starts at 0),
                so string[0] = "c", string[1] = "o", etc...
int(char c)     converts a character into its ASCII code,
                so if c = 'a', then int(c) = 97,
                and if c = 'c', the int(c) = 99, etc...
to_string(int i) converts an integer into a string. Function found in <string> library.
                Example, if i = 73, then to_string(i) = "73"
```

## string[n]

- What if you wanted to get a certain character from a string and make some calculations or decisions based on its value?
- But a “string[n]” is a character type. How can one use this data as a number instead??

# How/When to Use the Required Functions

## int (char c)

- One way to convert a char to an int is to use this device.
  - This is not the only way: you can use **static\_cast<int>** too
  - But I want you to learn an additional way of doing this...
- Note that in C++, int (char c) returns the integer value of the ASCII code of (char c).
  - Exampe, **int('2') = 50**
- So how can one work with that?

```
string.length() returns the length of a string, i
                so if string = "code", string.length() = 4.
string[n]       returns the nth character in the string (indexing starts at 0),
                so string[0] = "c", string[1] = "o", etc...
int(char c)     converts a character into its ASCII code,
                so if c = 'a', then int(c) = 97,
                and if c = 'c', the int(c) = 99, etc...
to_string(int i) converts an integer into a string. Function found in <string> library.
                Example, if i = 73, then to_string(i) = "73"
```



# How/When to Use the Required Functions

## `int (char c)`

- Look at the example of **`int('2') = 50`**
- Note that **`int('0') = 48`**
- Note again that **`int('2') – int('0') = 2`** !!!!
  - I now have the numerical character's integer value!
- So, if I have a (char c) that is a numerical character, and I do this:  
**`cnum = int (char c) – 48`**  
OR EVEN  
**`cnum = c – '0'`**  
I'll get the numerical value of (char c)!

```
string.length() returns the length of a string, i
                 so if string = "code", string.length() = 4.
string[n]        returns the nth character in the string (indexing starts at 0),
                 so string[0] = "c", string[1] = "o", etc...
int(char c)      converts a character into its ASCII code,
                 so if c = 'a', then int(c) = 97,
                 and if c = 'c', the int(c) = 99, etc...
to_string(int i) converts an integer into a string. Function found in <string> library.
                 Example, if i = 73, then to_string(i) = "73"
```

# Try it out...

- Try out this code snippet to test out this idea (and convince yourself that it works...!)

```
int c1, c2;
```

```
char c = '5';
```

```
c1 = int(c) - int('0');
```

```
c2 = c - '0';
```

```
cout << "c1 = int(c) - int('0') = " << c1 << endl;
```

```
cout << "c2 = c - '0' = " << c2 << endl;
```

***You should see 5 printed out on both lines***

# How/When to Use the Required Functions

```
string.length()    returns the length of a string, i
                  so if string = "code", string.length() = 4.
string[n]          returns the nth character in the string (indexing starts at 0),
                  so string[0] = "c", string[1] = "o", etc...
int(char c)        converts a character into its ASCII code,
                  so if c = 'a', then int(c) = 97,
                  and if c = 'c', the int(c) = 99, etc...
to_string(int i)   converts an integer into a string. Function found in <string> library.
                  Example, if i = 73, then to_string(i) = "73"
```

## to\_string(int i)

- Convenient for those times when you're switching between an integer calculation result and you want it to be a string...
- Pro-tip: some older compilers handle *to\_string()* strangely...
  - Sometimes they want you to use **std::to\_string()**, sometimes not.
  - Get into the habit of always compiling your code with the **-std=c++11** option when using g++: it's consistent in my experience.

# And a Word From Linux...

Some of you are having issues transferring files between CSIL & your computers. Here's the simplest way to do it:

- Open a Linux terminal window on your computer
- Issue the command:

```
scp <File on your computer> <username>@csil.cs.ucsb.edu:<dir. on CSIL>
```

EXAMPLE:

```
scp ./cs16/myProgram.cpp jimbo123@csil.cs.ucsb.edu:~jimbo/cs16/
```

- You have now copied your local file to a remote server (CSIL)!

- **OR**, to do it the **other** way around:

```
scp jimbo123@csil.cs.ucsb.edu:~jimbo/cs16/myProgram.cpp ./cs16/
```

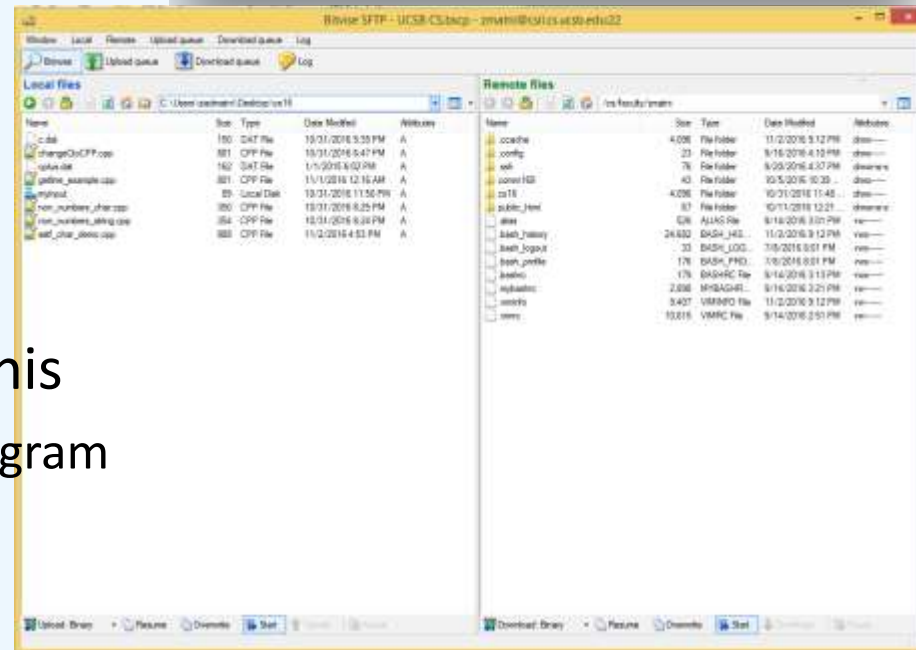
- You have now copied your local file to a remote server (CSIL)!

# Mac OS vs Windows OS

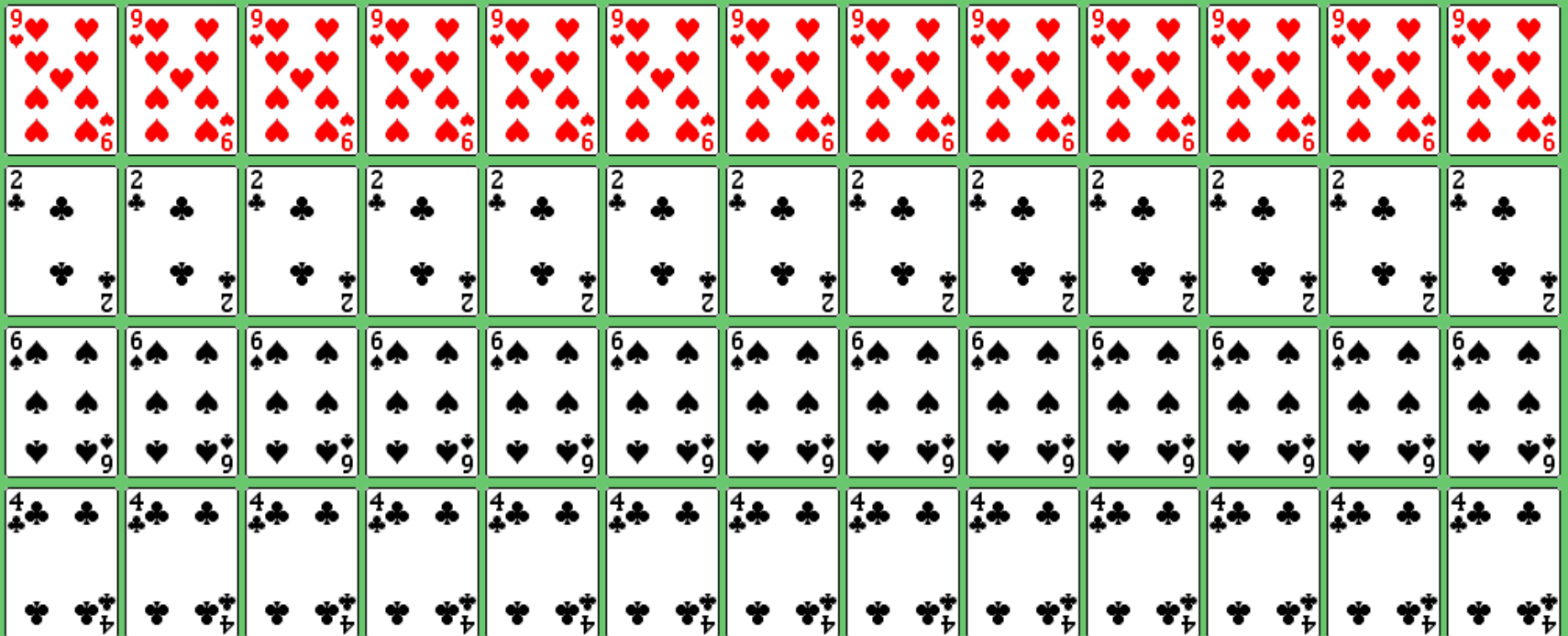
- Macs has a built-in Linux terminal app
- Windows 10 machines has one too
  - But you have to set it up first
  - Mac has a nicer “look and feel”, unfortunately
- Windows 8 & 7 DO NOT have this
  - Recommend you download a program like Bitwise SSH (works very well)
  - Can also be used on Windows 10



```
Kenny — bash — 78x20
Last login: Mon Apr 13 11:46:14 on ttys000
Kenny@MacBook-Pro:~$ mv ~/Documents/Test/TestFile-copy.rtf ~/Documents/Test2/TestFile-copy.rtf
```



# ARRAYS



# Introduction to Arrays

---

- An array is used to process a collection of data of the same type
  - Examples: A list of names  
A list of temperatures
- Why do we need arrays?
  - Imagine keeping track of 1000 test scores in memory!
    - How would you name all the variables?
    - How would you process each of the variables?

# Declaring an Array

- An array, named **score**, containing five variables of type int can be declared as

```
int score[5];
```

- This is like declaring 5 variables of type int:

```
score[0], score[1], ... , score[4]
```

- The value in brackets is called
  - A subscript or an index
- Note the **size** of the array is the **highest index value + 1**
  - Because indexing in C++ starts at 0, not 1



# Array Variable Types

---

- An array can have indexed variables of *any type* – they all just have to be the **SAME** type
- Use an indexed variable the same way an “ordinary” variable of the base type would be
- The square brackets [ ] hold the index
  - Can only be an integer number between 0 and (size – 1)
    - Can also be a variable that represents an integer number...

# Indexed Variable Assignment

---

- To assign a value to an indexed variable, use the assignment operator (just like with other variables):

```
int n = 2;  
score[n + 1] = 99;
```

- In this example, variable `score[3]` is assigned 99

# Loops And Arrays

- for-loops are commonly used to step through arrays

**Example:**

First index is 0

Last index is (size - 1)

```
for (i = 0; i < 5; i++)  
    cout << score[i] << " off by "  
        << (max - score[i]) << endl;
```

could display the difference between each score and the maximum score stored in an array

# Constants and Arrays

- Use **constants** to declare the size of an array
  - Using a constant allows your code to be easily altered for use on a smaller or larger set of data

## Example:

```
const int  NUMBER_OF_STUDENTS = 50;
int score[NUMBER_OF_STUDENTS];

...
for ( int i = 0; i < NUMBER_OF_STUDENTS; i++)
    cout << score[i] << " off by "
         << (max - score[i]) << endl;
```

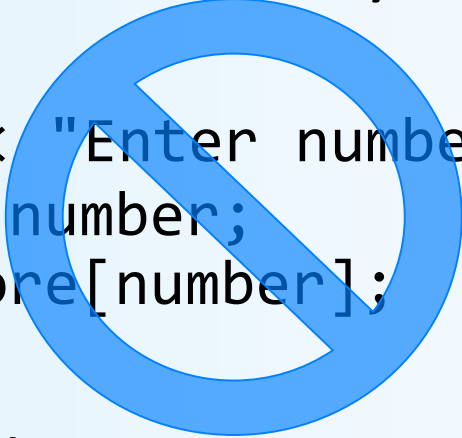
- To make this code work for any number of students, simply change the value of the constant in the 1<sup>st</sup> line...

# Variables and Declarations

- Most compilers ***do not allow*** the use of a variable to declare the size of an array

**Example**: 

```
cout << "Enter number of students: ";  
cin >> number;  
int score[number];
```



- This code is illegal on many C++ compilers
- Later we will take a look at **dynamic arrays** which do support this concept

# Array Declaration Syntax

---

- To declare an array, use the syntax:  
**Type\_Name Array\_Name[Declared\_Size];**
  - Type\_Name can be any type
  - Declared\_Size can be a constant to make your program more versatile
  - Example: `int scores[200]` or `char letters[26]`
- Once declared, an array contains the indexed variables:  
**Array\_Name[0] to Array\_Name[Declared\_Size -1]**

# Computer Memory

---

- Computer memory consists of numbered locations in bytes (i.e. 8 bits at a time)
  - A byte's number is its *address*
- A simple variable is stored in consecutive bytes
  - The number of bytes depends on the variable's type
  - Example: **int** will have fewer bytes than **long int** types
- A variable's address is the address of its **first** byte

# Arrays and Memory

- Declaring the array `int a[6]`
  - Reserves memory for six variables of type **int**
  - The variables are stored one after another
  - The address of **a[0]** is remembered
    - The addresses of the other indexed variables is not remembered (no need to)
  - To determine the address of **a[3]**
    - Start at **a[0]**
    - Count past enough memory for three integers to find **a[3]**





# Array Index Out of Range

---

- A common error is using a nonexistent index
  - Index values for `int a[6]` are the values  
0 through 5
  - An index value that's not allowed by the array declaration is *out of range*
  - Using an out of range index value does not produce an error message!

# Out of Range Problems

- If an array is declared as: `int a[6];`  
and an integer is declared as: `int i = 7;`
- Executing the statement: `a[i] = 238;`  
causes...
  - The computer to calculate the address of the illegal `a[7]`
    - This address could be where some other variable is stored
  - The value 238 is stored at the address calculated for `a[7]`
  - No warning is given!
- *This is bad practice! Keep track of your arrays!*

# Initializing Arrays

- To initialize an array when it is declared
  - The values for the indexed variables are enclosed in braces and separated by commas

- Example: `int children[3] = {2, 12, 1};`

Is equivalent to:

```
int children[3];
children[0] = 2;
children[1] = 12;
children[2] = 1;
```

# Default Values

- If too few values are listed in an initialization statement
  - The listed values are used to initialize the first of the indexed variables
  - The remaining indexed variables are initialized to a **zero** of the base type
- Example: `int a[10] = {5, 5};`  
initializes `a[0]` and `a[1]` to 5  
and `a[2]` through `a[9]` to 0

**NOTE**: This is called an *extended initializer list* and it only works in the latest versions of C++ compilers. So make sure you compile with the `-std=c++11` option when using g++.

# Is this OK?

```
int num[ ] = {0, 0, 0};
```

- *When an array is initialized*, C++ allows you to leave the square brackets empty [ ]
- In this case, the compiler will assume automatically a size for the array that matches the number of values included between the braces { }
- So, the example given here makes the array **num** automatically size 3.
- This shortcut is ok for initializing small arrays, but imagine if **num** is size 300 instead...!

# Un-initialized Arrays

---

- If no values are listed in the array declaration, some compilers might initialize each variable to a zero of the base type
  - DO NOT DEPEND ON THIS!!
- g++ does not do this, FYI...
- *Lesson learned:*  
Initialize your array variables for the same reasons you initialize your “regular” variables

# Range-Based For Loops

- C++11 includes a new type of for loop: **the range-based for-loop**, that simplifies iteration over every element in an array. The syntax is shown below:

```
for (datatype varname : array)
{
    // varname is successively set to each
    // element in the array
}
```



# Range-Based For Loop Example

---

- The following code outputs: **2 4 6 8**

```
int arr[ ] = {2, 4, 6, 8};  
for (int x : arr) {  
    cout << x;  
    cout << " "; }  
}
```



# Arrays in Functions

- Indexed variables can be arguments to functions
- Example:

If a program contains these declarations:

```
int i, n, a[10];  
void my_function(int n);
```

Variables a[0] through a[9] are of type int, making these calls legal:

```
my_function( a[0] );  
my_function( a[3] );  
my_function( a[i] );
```

# Arrays as Function Arguments

---

- A formal parameter can be for an entire array
- Such a parameter is called an array parameter
  - It is not a call-by-value parameter
  - It is not a call-by-reference parameter
  - Although, array parameters behave much like call-by-reference parameters

# Array Parameter Declaration

---

- An array parameter is indicated using empty brackets in the parameter list such as

```
void fill_up(int a[], int size);
```

# Function Calls With Arrays

---

- If function **fill\_up** is declared in this way:  

```
void fill_up(int a[], int size);
```
- and array **score** is declared this way:  

```
int score[5], number_of_scores;
```
- **fill\_up** is called in this way:  

```
fill_up(score, number_of_scores);
```

## Function with an Array Parameter

---

### Function Declaration

```
void fill_up(int a[], int size);  
//Precondition: size is the declared size of the array a.  
//The user will type in size integers.  
//Postcondition: The array a is filled with size integers  
//from the keyboard.
```

### Function Definition

```
//Uses iostream:  
void fill_up(int a[], int size)  
{  
    using namespace std;  
    cout << "Enter " << size << " numbers:\n";  
    for (int i = 0; i < size; i++)  
        cin >> a[i];  
    size--;  
    cout << "The last array index used is " << size << endl;  
}
```

# Function Call Details

- A formal parameter is identified as an array parameter by the [ ]'s with no index expression

*Function Description*

```
void fill_up(int a[ ], int size);
```

- An array argument does not use the [ ]'s

*Function Call*

```
fill_up(score, number_of_scores);
```

- Note that the values of the indexed variables can be changed by the function



# Array Argument Details

---

- What does the computer know about an array?
  - The base type
  - The address of the first indexed variable
  - The number of indexed variables
- What does a function know about an array argument?
  - The base type
  - The address of the first indexed variable

# Array Parameter Considerations

---

- Because a function does not know the size of an array argument...
  - The programmer should include a formal parameter that specifies the size of the array
  - The function can process arrays of various sizes
    - Function **fill\_up** from Display 7.4 on pg. 392 of the textbook can be used to fill an array of any size:

```
fill_up(score, 5);  
fill_up(time, 10);
```

# But... IS there a way to CALCULATE the Size of an Array?

---

- Yes
- More on that later...
- For now, get used to the idea of passing the size of an array into a function that has the array as argument.

# const Modifier

---

- Array parameters allow a function to change the values stored in the array argument
- If a function *should not change* the values of the array argument, use the modifier **const**
- An array parameter modified with **const** is 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** is used in *both* the function declaration and definition to modify the array parameter
  - The compiler will issue an error if you write code that changes the values stored in the array parameter

# Function Calls and const

---

- If a function with a constant array parameter calls another function using the **const** array parameter as an argument...
  - The called function must use a constant array parameter as a placeholder for the array
  - The compiler will issue an error if a function is called that does not have a **const** array parameter to accept the array argument

# const Parameters Example

```
double compute_average(int a[ ], int size);  
  
void show_difference(const int a[ ], int size)  
{  
    double average = compute_average(a, size);  
    ...  
}
```

- `compute_average` has no constant array parameter
- This code generates an error message because `compute_average` could change the array parameter

# Returning An Array

---

- Recall that functions can return a value of type int, double, char, ..., or a class type
- **Functions cannot return arrays**
- We learn later how to return  
a pointer to an array





# TO DOs

---

- Homework #12 due Tuesday 11/1
- Lab #6
  - Due Friday, 11/4, at noon

**</LECTURE>**