

APS105: Lecture 16

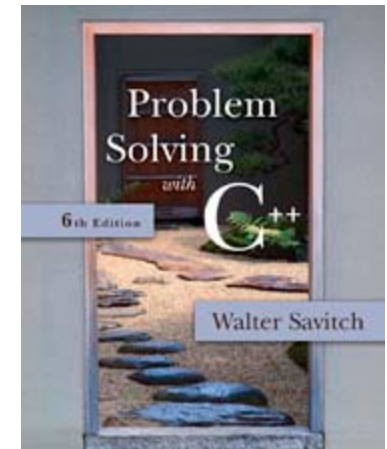
Wael Aboelsaadat

wael@cs.toronto.edu

<http://ccnet3.utoronto.ca/20079/aps105h1f/>

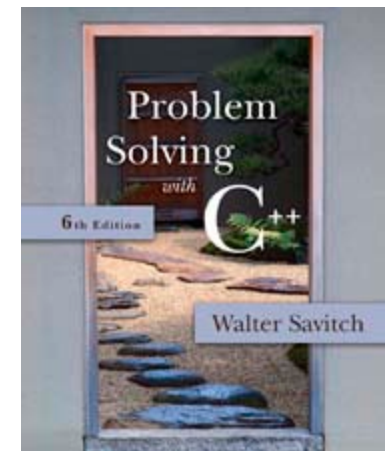
Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley

Download the code shown in lecture from course website:
Handouts → Lectures Source Code - Wael



Chapter 7

Arrays



Overview

7.1 Introduction to Arrays

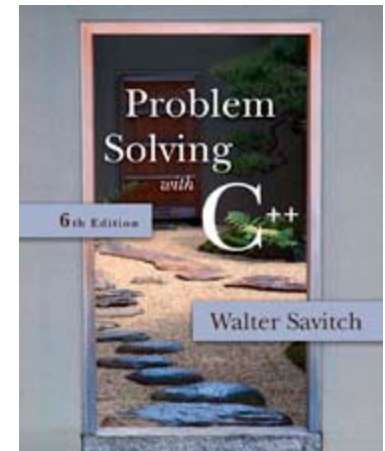
7.2 Arrays in Functions

7.3 Programming with Arrays

7.4 Multidimensional Arrays

7.1

Introduction to 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 5 test scores, or 100, or 1000 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
 - An index

The Array Variables

- The variables making up the array are referred to as
 - Indexed variables
 - Subscripted variables
 - Elements of the array
- The number of indexed variables in an array is the declared size, or size, of the array
 - The largest index is one less than the size
 - The first index value is zero

Array Variable Types

- An array can have indexed variables of any type
- All indexed variables in an array are of the same type
 - This is the base type of the array
- An indexed variable can be used anywhere an ordinary variable of the base type is used

Using [] With Arrays

- In an array declaration, []'s enclose the size of the array such as this array of 5 integers:

```
int score [5];
```
- When referring to one of the indexed variables, the []'s enclose a number identifying one of the indexed variables
 - score[3] is one of the indexed variables
 - The value in the []'s can be any expression that evaluates to one of the integers 0 to (size -1)

Indexed Variable Assignment

- To assign a value to an indexed variable, use the assignment operator:

```
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

First index is 0

Last index is (size - 1)

- Example:

```
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

Display 7.1

Constants and Arrays

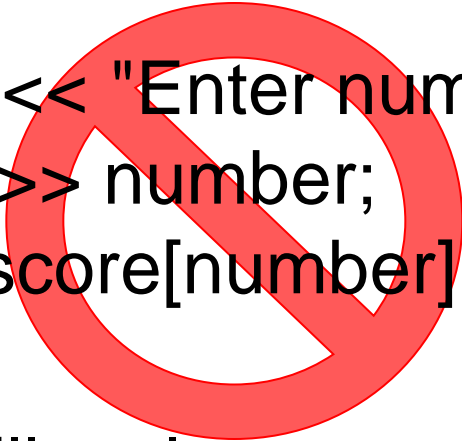
- Use constants to declare the size of an array
 - Using a constant allows your code to be easily altered for use on a smaller or larger set of data
 - Example:

```
const int NUMBER_OF_STUDENTS = 50;
int score[NUMBER_OF_STUDENTS];
...
for ( i = 0; i < NUMBER_OF_STUDENTS;
i++)
    cout << score[i] << " off by "
        << (max - score[i]) << endl;
```
 - Only the value of the constant must be changed to make this code work for any number of students

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 compilers

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
- Once declared, the array consists of the indexed variables:
Array_Name[0] to Array_Name[Declared_Size -1]

Computer Memory

- Computer memory consists of numbered locations called bytes
 - 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
- 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
 - To determine the address of `a[3]`
 - Start at `a[0]`
 - Count past enough memory for three integers to find `a[3]`

Display 7.2

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 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!

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

Un-initialized Arrays

- If no values are listed in the array declaration, some compilers will initialize each variable to a zero of the base type
 - DO NOT DEPEND ON THIS!

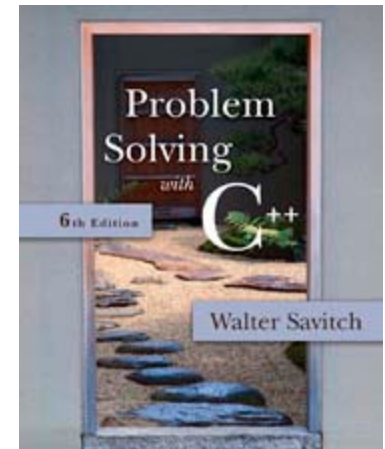
Section 7.1 Conclusion

- Can you
 - Describe the difference between `a[4]` and `int a[5]`?
 - Show the output of

```
char symbol[3] = {'a', 'b', 'c'};  
for (int index = 0; index < 3; index++)  
    cout << symbol[index];
```

7.2

Arrays in Functions



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 ] );
```

Display 7.3

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
 - 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);
```

```
//Reads in 5 scores and shows how much each
//score differs from the highest score.
#include <iostream>

int main()
{
    using namespace std;
    int i, score[5], max;

    cout << "Enter 5 scores:\n";
    cin >> score[0];
    max = score[0];
    for (i = 1; i < 5; i++)
    {
        cin >> score[i];
        if (score[i] > max)
            max = score[i];
        //max is the largest of the values score[0],..., score[i].
    }

    cout << "The highest score is " << max << endl
         << "The scores and their\n"
         << "differences from the highest are:\n";
    for (i = 0; i < 5; i++)
        cout << score[i] << " off by "
             << (max - score[i]) << endl;

    return 0;
}
```

Sample Dialogue

```
Enter 5 scores:
5 9 2 10 6
The highest score is 10
The scores and their
differences from the highest are:
5 off by 5
9 off by 1
2 off by 8
10 off by 0
6 off by 4
```

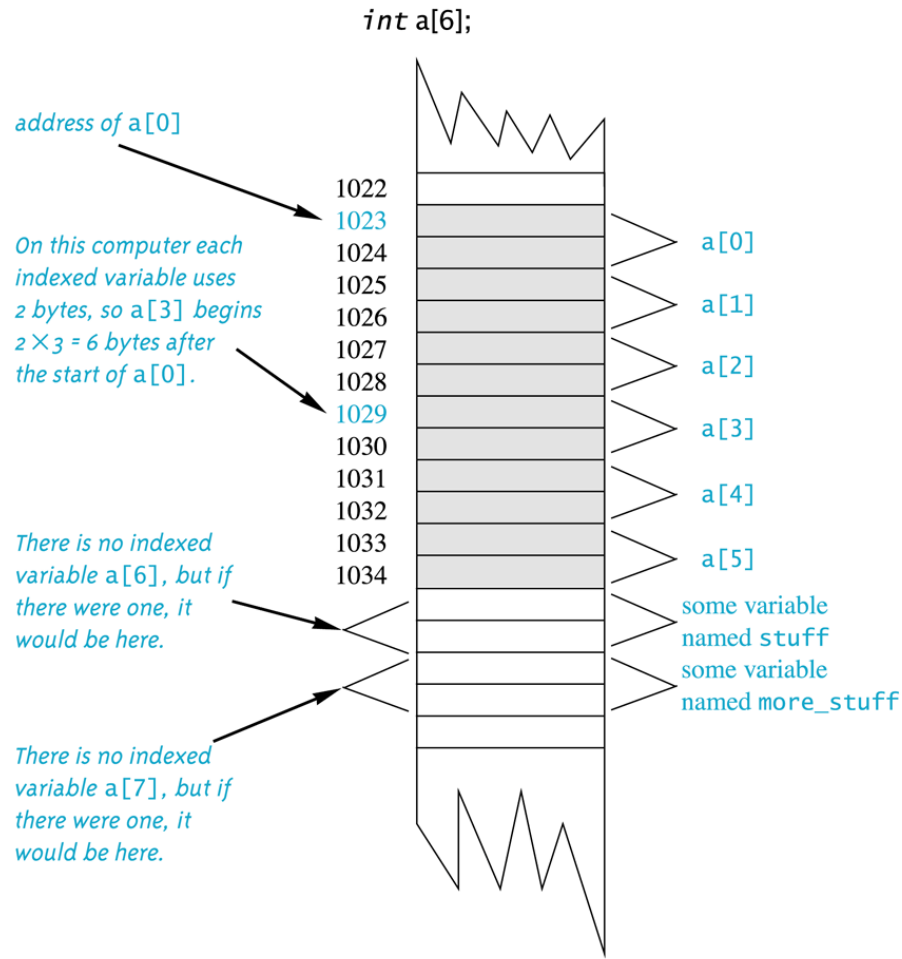
Display 7.1



Display 7.2



An Array in Memory



```
//Illustrates the use of an indexed variable as an argument.
//Adds 5 to each employee's allowed number of vacation days.
#include <iostream>

const int NUMBER_OF_EMPLOYEES = 3;

int adjust_days(int old_days);
//Returns old_days plus 5.

int main()
{
    using namespace std;
    int vacation[NUMBER_OF_EMPLOYEES], number;

    cout << "Enter allowed vacation days for employees 1"
         << " through " << NUMBER_OF_EMPLOYEES << ":\n";
    for (number = 1; number <= NUMBER_OF_EMPLOYEES; number++)
        cin >> vacation[number-1];

    for (number = 0; number < NUMBER_OF_EMPLOYEES; number++)
        vacation[number] = adjust_days(vacation[number]);

    cout << "The revised number of vacation days are:\n";
    for (number = 1; number <= NUMBER_OF_EMPLOYEES; number++)
        cout << "Employee number " << number
             << " vacation days = " << vacation[number-1] << endl;

    return 0;
}

int adjust_days(int old_days)
{
    return (old_days + 5);
}
```

Sample Dialogue

Enter allowed vacation days for employees 1 through 3:

10 20 5

The revised number of vacation days are:

Employee number 1 vacation days = 15

Employee number 2 vacation days = 25

Employee number 3 vacation days = 10

Display 7.3

