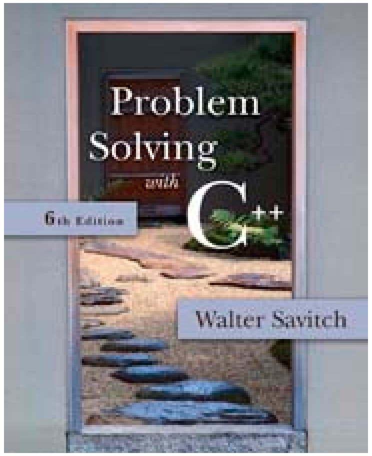
APS105: Lecture 23

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



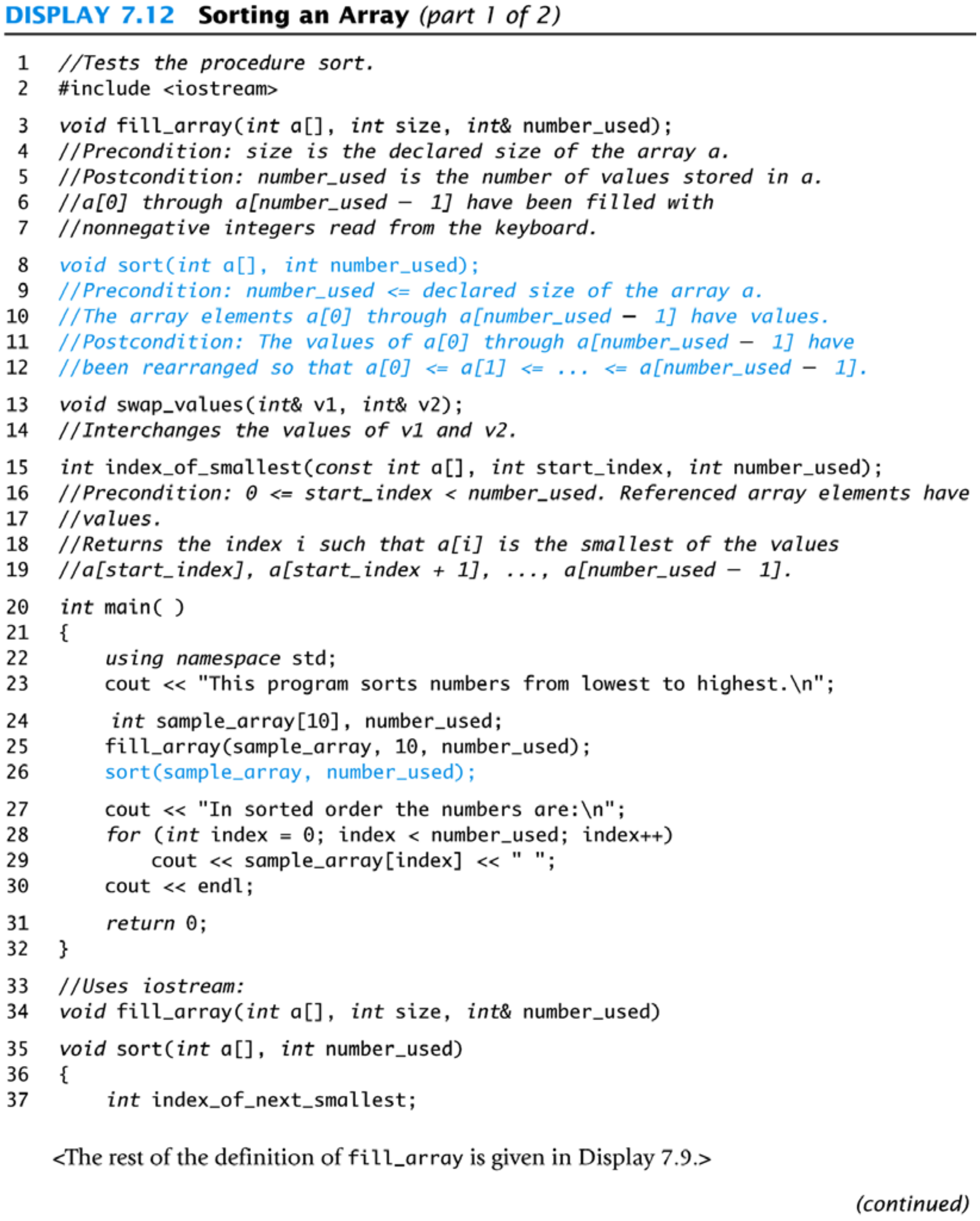
Selection Sort Algorithm

- 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
  - 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

Display 7.11 **Display 7.12 (1-3)**

Display 7.12 (1/2)

Back Next



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

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

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.

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

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

int index_of_smallest(const int a[], int start_index, int number_used);
//Precondition: 0 <= start_index < number_used. Referenced array elements have
//values.
//Returns the index i such that a[i] is the smallest of the values
//a[start_index], a[start_index + 1], ..., a[number_used − 1].

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

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

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

    return 0;
}

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

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

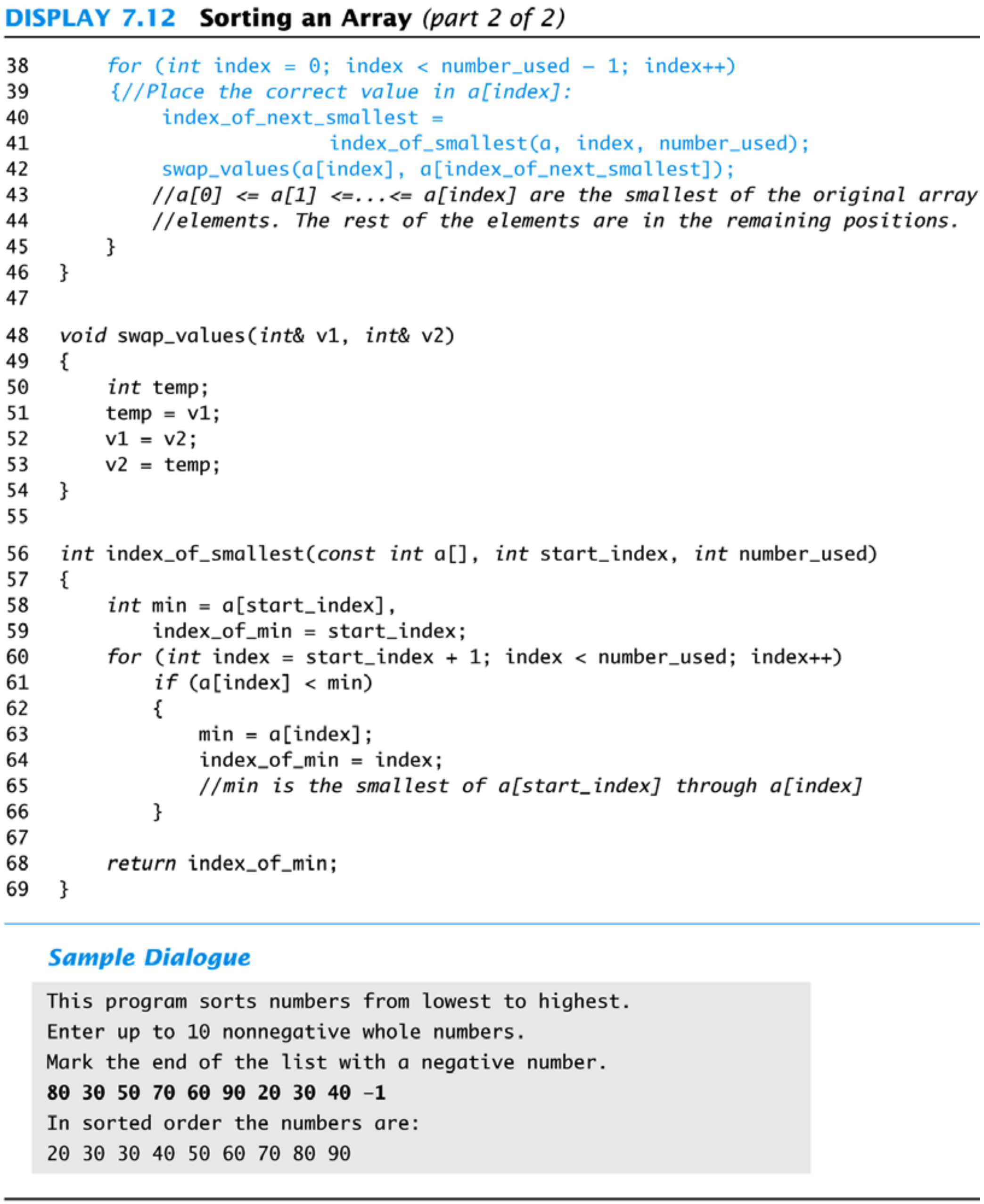
<The rest of the definition of fill_array is given in Display 7.9.>

(continued)

Display 7.12 (2/2)

Back

Next



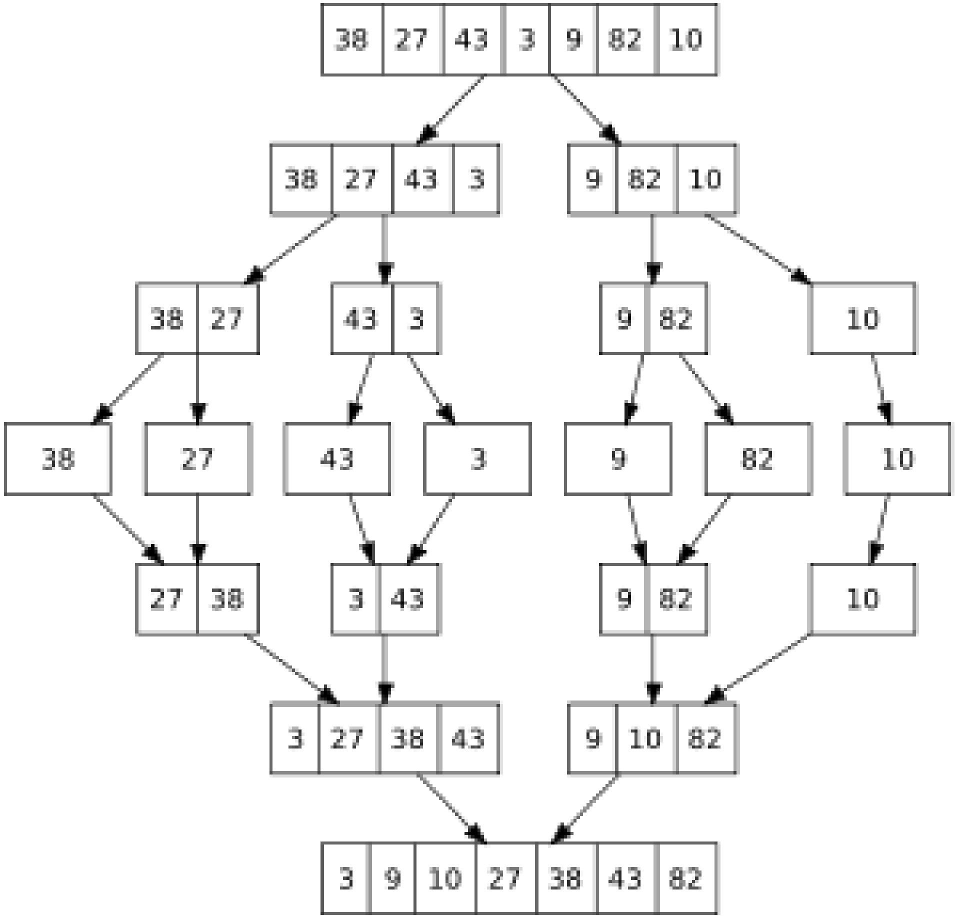
DISPLAY 7.12 Sorting an Array *(part 2 of 2)*

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

void swap_values(int& v1, int& v2)
{
    int temp;
    temp = v1;
    v1 = v2;
    v2 = temp;
}

int index_of_smallest(const int a[], int start_index, int number_used)
{
    int min = a[start_index],
        index_of_min = start_index;
    for (int index = start_index + 1; index < number_used; index++)
        if (a[index] < min)
        {
            min = a[index];
            index_of_min = index;
            //min is the smallest of a[start_index] through a[index]
        }

    return index_of_min;
}
```

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
```

Merge Sort Algorithm

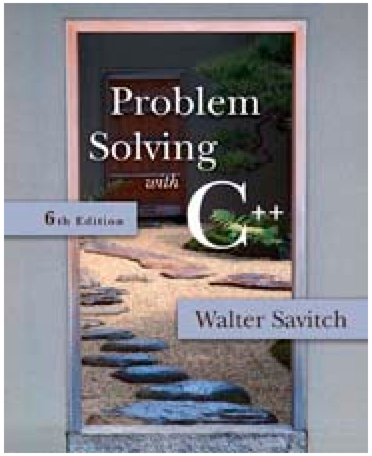


From Wikipedia

http://www.iste.uni-stuttgart.de/ps/Ploedereder/sorter/sortanimation2.html

Chapter 9

Pointers and Dynamic Arrays



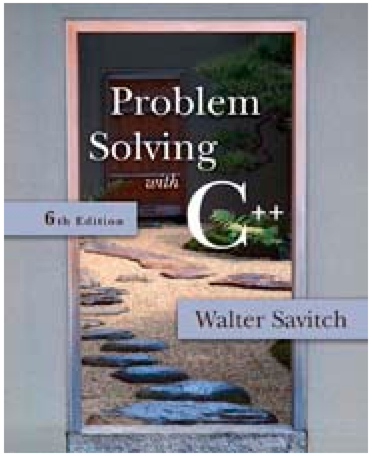
Overview

9.1 Pointers

9.2 Dynamic Arrays

9.1

Pointers



Problem Solving with C++

6th Edition

Walter Savitch

Pointers

- A pointer is the memory address of a variable
- Memory addresses can be used as names for variables
  - If a variable is stored in three memory locations, the address of the first can be used as a name for the variable.
  - When a variable is used as a call-by-reference argument, its address is passed

Pointers Tell
Where To Find A Variable

- An address used to tell where a variable is stored in memory is a pointer
  - Pointers "point" to a variable by telling where the variable is located

Declaring Pointers

- Pointer variables must be declared to have a pointer type
  - Example: To declare a pointer variable p that can "point" to a variable of type double:

    ```
    double  *p;
    ```
  - The asterisk identifies p as a pointer variable

Multiple Pointer Declarations

- To declare multiple pointers in a statement, use the asterisk before each pointer variable
  - Example:

```
int *p1, *p2, v1, v2;
```

    p1 and p2 point to variables of type int
    v1 and v2 are variables of type int

The address of Operator

- The & operator can be used to determine the address of a variable which can be assigned to a pointer variable
  - Example: p1 = &v1;

p1 is now a pointer to v1

v1 can be called v1 or "the variable pointed to

by p1"

The Dereferencing Operator

- C++ uses the * operator in yet another way with pointers
  - The phrase "The variable pointed to by p" is translated into C++ as *p
  - Here the * is the dereferencing operator
    - p is said to be dereferenced

A Pointer Example

-
  ```
  v1 = 0;
  p1 = &v1;
  *p1 = 42;
  cout << v1 << endl;
  cout << *p1 << endl;
  ```

  output:

  ```
             42
             42
  ```

v1 and *p1 now refer to the same variable