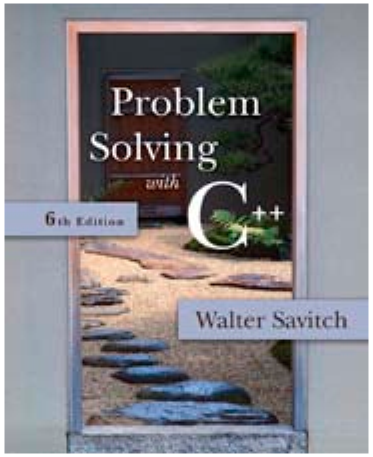
APS105: Lecture 31B

Wael Aboelsaadat

wael@cs.toronto.edu

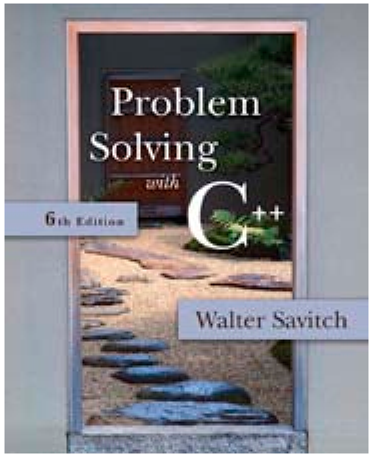
http://ccnet3.utoronto.ca/20079/aps105h1f/

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



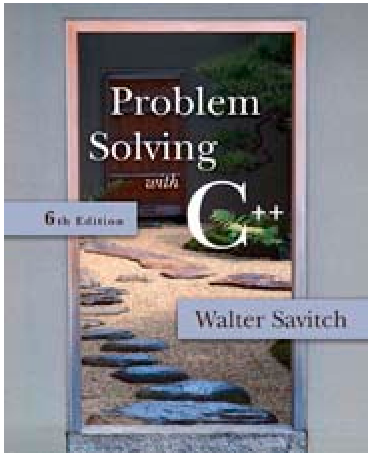
Chapter 9

Pointers and Dynamic Arrays



9.2

Dynamic Arrays



Dynamic Arrays

- A dynamic array is an array whose size is determined when the program is running, not when you write the program

Pointer Variables and Array Variables

- Array variables are actually pointer variables that point to the first indexed variable
  - Example:
    ```
    int  a[10];
    typedef int* IntPtr;
     IntPtr p;
    ```
    - Variables a and p are the same kind of variable
- Since a is a pointer variable that points to a[0],

  ```
  p = a;
  ```

  causes p to point to the same location as a

Pointer Variables As Array Variables

- Continuing the previous example:
  Pointer variable p can be used as if it were an array variable
- Example: p[0], p[1], …p[9]
  are all legal ways to use p
- Variable a can be used as a pointer variable except the pointer value in a cannot be changed
  - This is not legal:

```
IntPtr p2;
 … // p2 is assigned a value
a = p2  // attempt to change a
```

Display 9.4

Creating Dynamic Arrays

- Normal arrays require that the programmer determine the size of the array when the program is written
  - What if the programmer estimates too large?
    - Memory is wasted
  - What if the programmer estimates too small?
    - The program may not work in some situations
- Dynamic arrays can be created with just the right size while the program is running

Creating Dynamic Arrays

- Dynamic arrays are created using the new operator
  - Example: To create an array of 10 elements of type double:

```
typedef double* DoublePtr;
DoublePtr d;
d = new double[10];
```

This could be an integer variable!

- d can now be used as if it were an ordinary array!

Dynamic Arrays (cont.)

- Pointer variable d is a pointer to d[0]
- When finished with the array, it should be deleted to return memory to the freestore
  - Example: delete [ ] d;
    - The brackets tell C++ a dynamic array is being deleted so it must check the size to know how many indexed variables to remove
    - Forgetting the brackets, is not illegal, but would tell the computer to remove only one variable

Display 9.5 (1)

Display 9.5 (2)

Pointer Arithmetic (Optional)

- Arithmetic can be performed on the addresses contained in pointers
  - Using the dynamic array of doubles, d, declared previously, recall that d points to d[0]
  - The expression d+1 evaluates to the address of d[1] and d+2 evaluates to the address of d[2]
    - Notice that adding one adds enough bytes for one variable of the type stored in the array

Pointer Arthmetic Operations

- You can add and subtract with pointers
  - The ++ and - - operators can be used
  - Two pointers of the same type can be subtracted to obtain the number of indexed variables between
    - The pointers should be in the same array!
  - This code shows one way to use pointer arithmetic:

```
for (int i = 0; i < array_size; i++)
    cout << *(d + i) << "  " ;
    // same as cout << d[i] << "  " ;
```

Multidimensional Dynamic Arrays

- To create a 3x4 multidimensional dynamic array
  - View multidimensional arrays as arrays of arrays
  - First create a one-dimensional dynamic array
    - Start with a new definition:

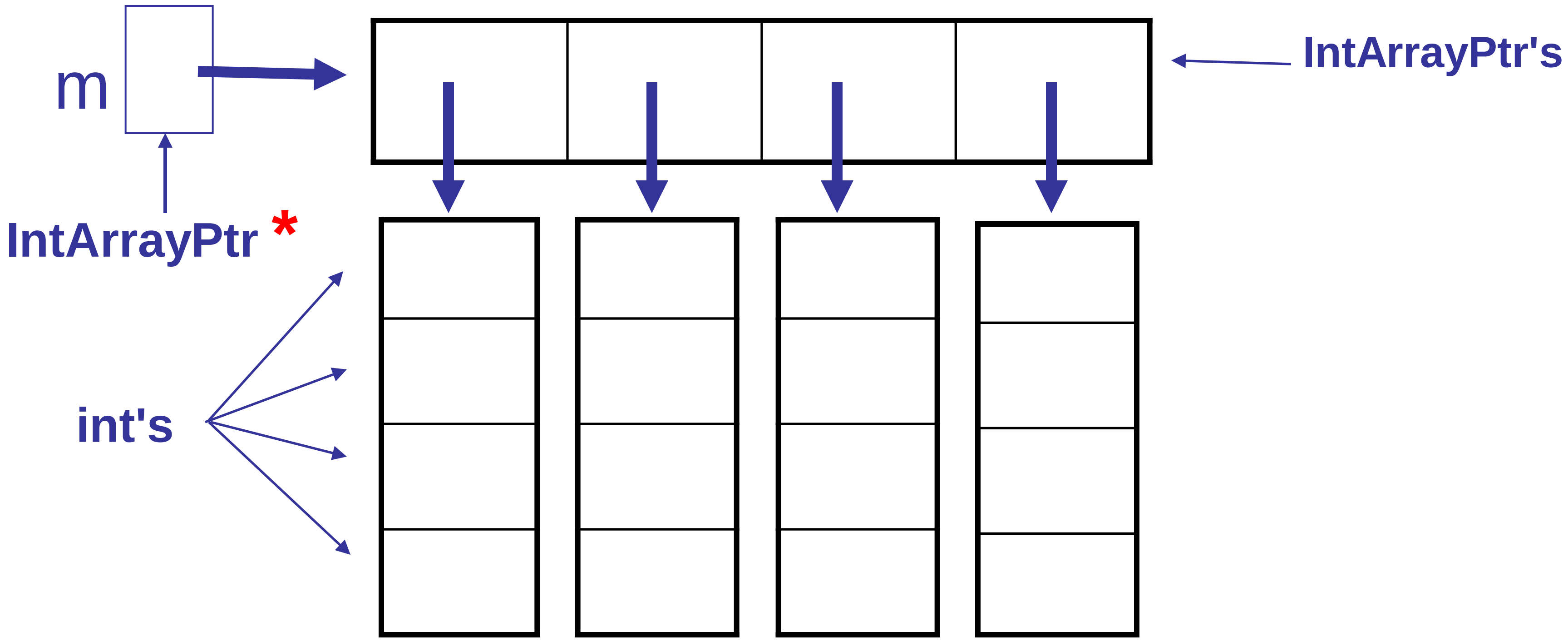
      ```
      typedef int* IntArrayPtr;
      ```
    - Now create a dynamic array of pointers named m:

      ```
      IntArrayPtr *m = new IntArrayPtr[3];
      ```
  - For each pointer in m, create a dynamic array of int's

    -
      ```
      for (int i = 0; i<3; i++)
          m[i] = new int[4];
      ```

A Multidimensial Dynamic Array

- The dynamic array created on the previous slide could be visualized like this:



Deleting Multidimensional Arrays

- To delete a multidimensional dynamic array
  - Each call to new that created an array must have a corresponding call to delete[ ]
  - Example: To delete the dynamic array created on a previous slide:

```
for ( i = 0; i < 3; i++)
    delete [ ] m[i]; //delete the arrays of 4 int's
delete [ ] m; // delete the array of IntArrayPtr's
```

Display 9.6 (1) Display 9.6 (2)

Display 9.4

Back Next

Arrays and Pointer Variables

```
//Program to demonstrate that an array variable is a kind of pointer variable.
#include <iostream>
using namespace std;

typedef int* IntPtr;

int main( )
{
    IntPtr p;
    int a[10];
    int index;

    for (index = 0; index < 10; index++)
        a[index] = index;

    p = a;

    for (index = 0; index < 10; index++)
        cout << p[index] << " ";
    cout << endl;

    for (index = 0; index < 10; index++)
        p[index] = p[index] + 1;

    for (index = 0; index < 10; index++)
        cout << a[index] << " ";
    cout << endl;

    return 0;
}
```

Note that changes to the array p are also changes to the array a.

Output

```
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

Display 9.5 (1/2)

Back Next

DISPLAY 9.5 A Dynamic Array *(part 1 of 2)*

```
//Sorts a list of numbers entered at the keyboard.
#include <iostream>
#include <cstdlib>
#include <cstddef>

typedef int* IntArrayPtr;

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

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

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

     int array_size;
     cout << "How many numbers will be sorted? ";
     cin >> array_size;

     IntArrayPtr a;
     a = new int[array_size];

     fill_array(a, array_size);
     sort(a, array_size);

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

     delete [] a;

     return 0;
}
```

Ordinary array parameters

The dynamic array a *is used like an ordinary array.*

(continued)

Display 9.5 (2/2)

Back Next

DISPLAY 9.5 A Dynamic Array *(part 2 of 2)*

```
//Uses the library iostream:
void fill_array(int a[], int size)
{
     using namespace std;
     cout << "Enter " << size << " integers.\n";
     for (int index = 0; index < size; index++)
         cin >> a[index];
}

void sort(int a[], int size)
```

<Any implementation of sort may be used. This may or may not require some additional function definitions. The implementation need not even know that sort will be called with a dynamic array. For example, you can use the implementation in Display 7.12 (with suitable adjustments to parameter names).>

Display 9.6 (1/2)

Back Next

A Two-Dimensional Dynamic Array (*part 1 of 2*)

```
#include <iostream>
using namespace std;

typedef int* IntArrayPtr;

int main( )
{
    int d1, d2;
    cout << "Enter the row and column dimensions of the array:\n";
    cin >> d1 >> d2;

    IntArrayPtr *m = new IntArrayPtr[d1];
    int i, j;
    for (i = 0; i < d1; i++)
        m[i] = new int[d2];
    //m is now a d1 by d2 array.

    cout << "Enter " << d1 << " rows of "
         << d2 << " integers each:\n";
    for (i = 0; i < d1; i++)
        for (j = 0; j < d2; j++)
            cin >> m[i][j];

    cout << "Echoing the two-dimensional array:\n";
    for (i = 0; i < d1; i++)
    {
        for (j = 0; j < d2; j++)
            cout << m[i][j] << " ";
        cout << endl;
    }
```

Display 9.6 (2/2)

Back Next

A Two-Dimensional Dynamic Array (*part 2 of 2*)

```
    for (i = 0; i < d1; i++)
        delete[] m[i];
    delete[] m;

    return 0;
}
```

Note that there must be one call to `delete []` *for each call to new that created an array. (These calls to* `delete []` *are not really needed since the program is ending, but in another context it could be important to include them.)*

Sample Dialogue

```
Enter the row and column dimensions of the array:
3 4
Enter 3 rows of 4 integers each:
1 2 3 4
5 6 7 8
9 0 1 2
Echoing the two-dimensional array:
1 2 3 4
5 6 7 8
9 0 1 2
```