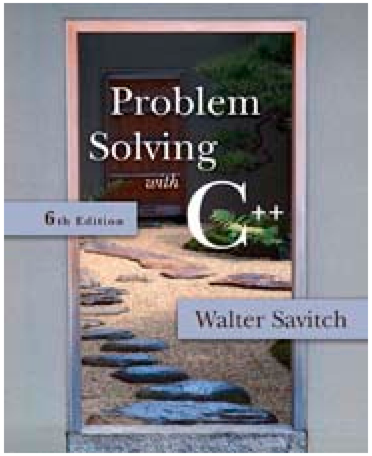
APS105: Lecture 24

Wael Aboelsaadat

wael@cs.toronto.edu

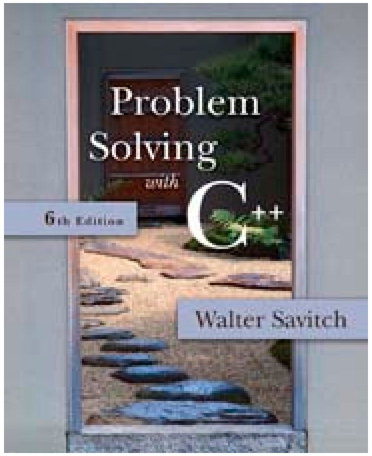
http://ccnet3.utoronto.ca/20079/aps105h1f/

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



Chapter 9

Pointers and Dynamic Arrays



PEARSON
Addison
Wesley

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

Pointer Assignment

- The assignment operator = is used to assign the value of one pointer to another
  - Example: If p1 still points to v1 (previous slide)

    then

    p2 = p1;

    causes *p2, *p1, and v1 all to name

    the same variable

Caution! Pointer Assignments

- Some care is required making assignments to pointer variables
  - p1= p3; // changes the location that p1 "points" to
  - *p1 = *p3; // changes the value at the location that

    // p1 "points" to

Display 9.1

The new Operator

- Using pointers, variables can be manipulated even if there is no identifier for them
  - To create a pointer to a new "nameless" variable of type int:

    ```
    p1 = new int;
    ```

  - The new variable is referred to as *p1
  - *p1 can be used anyplace an integer variable can

    ```
    cin >> *p1;
    *p1 = *p1 + 7;
    ```

Dynamic Variables

- Variables created using the new operator are called dynamic variables
  - Dynamic variables are created and destroyed while the program is running
  - Additional examples of pointers and dynamic variables are shown in **Display 9.2**

    An illustration of the code in Display 9.2 is seen in **Display 9.3**

new and Class Types

- Using operator new with class types calls a constructor as well as allocating memory
  - If MyType is a class type, then

```
MyType *myPtr; // creates a pointer to a
                          // variable of type MyType
 myPtr = new MyType;
              // calls the default constructor

  myPtr  = new MyType (32.0, 17);
               // calls  Mytype(double, int);
```

Basic Memory Management

- An area of memory called the freestore is reserved for dynamic variables
  - New dynamic variables use memory in the freestore
  - If all of the freestore is used, calls to new will fail
- Unneeded memory can be recycled
  - When variables are no longer needed, they can be deleted and the memory they used is returned to the freestore

The delete Operator

- When dynamic variables are no longer needed, delete them to return memory to the freestore
  - Example:

    ```
    delete p;
    ```

    The value of p is now undefined and the memory used by the variable that p pointed to is back in the freestore

Dangling Pointers

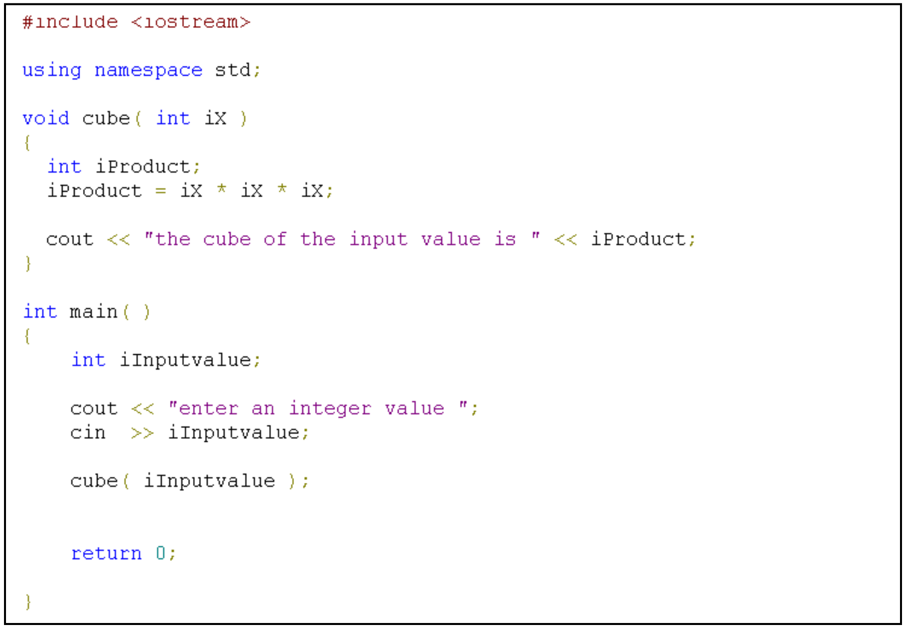
- Using delete on a pointer variable destroys the dynamic variable pointed to
- If another pointer variable was pointing to the dynamic variable, that variable is also undefined
- Undefined pointer variables are called dangling pointers
  - Dereferencing a dangling pointer (*p) is usually disasterous

Variable Types & Lifetime

- Automatic variables
- Dynamic Variables
- Global variables

Automatic Variables

- Variables declared in a function are created by C++ and destroyed when the function ends
  - These are called automatic variables because their creation and destruction is controlled automatically



```
#include <iostream>

using namespace std;

void cube( int iX )
{
  int iProduct;
  iProduct = iX * iX * iX;

  cout << "the cube of the input value is " << iProduct;
}

int main( )
{
    int iInputvalue;

    cout << "enter an integer value ";
    cin  >> iInputvalue;

    cube( iInputvalue );


    return 0;

}
```

Dynamic Variables

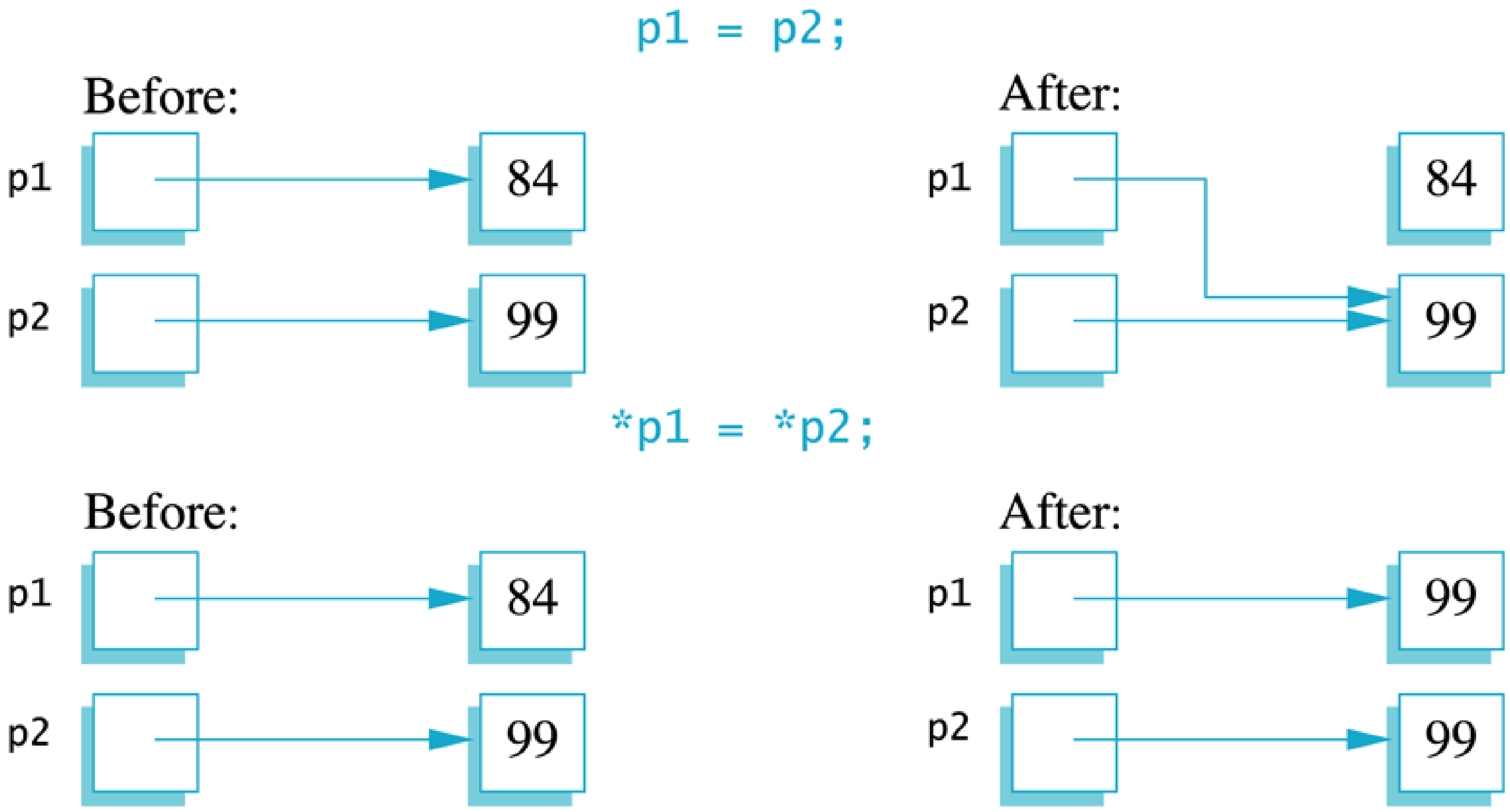
- The programmer manually controls creation and destruction of pointer variables with operators new and delete

Global Variables

- Variables declared outside any function definition are global variables
  - Global variables are available to all parts of a program
  - Global variables are not recommended as good programming practice

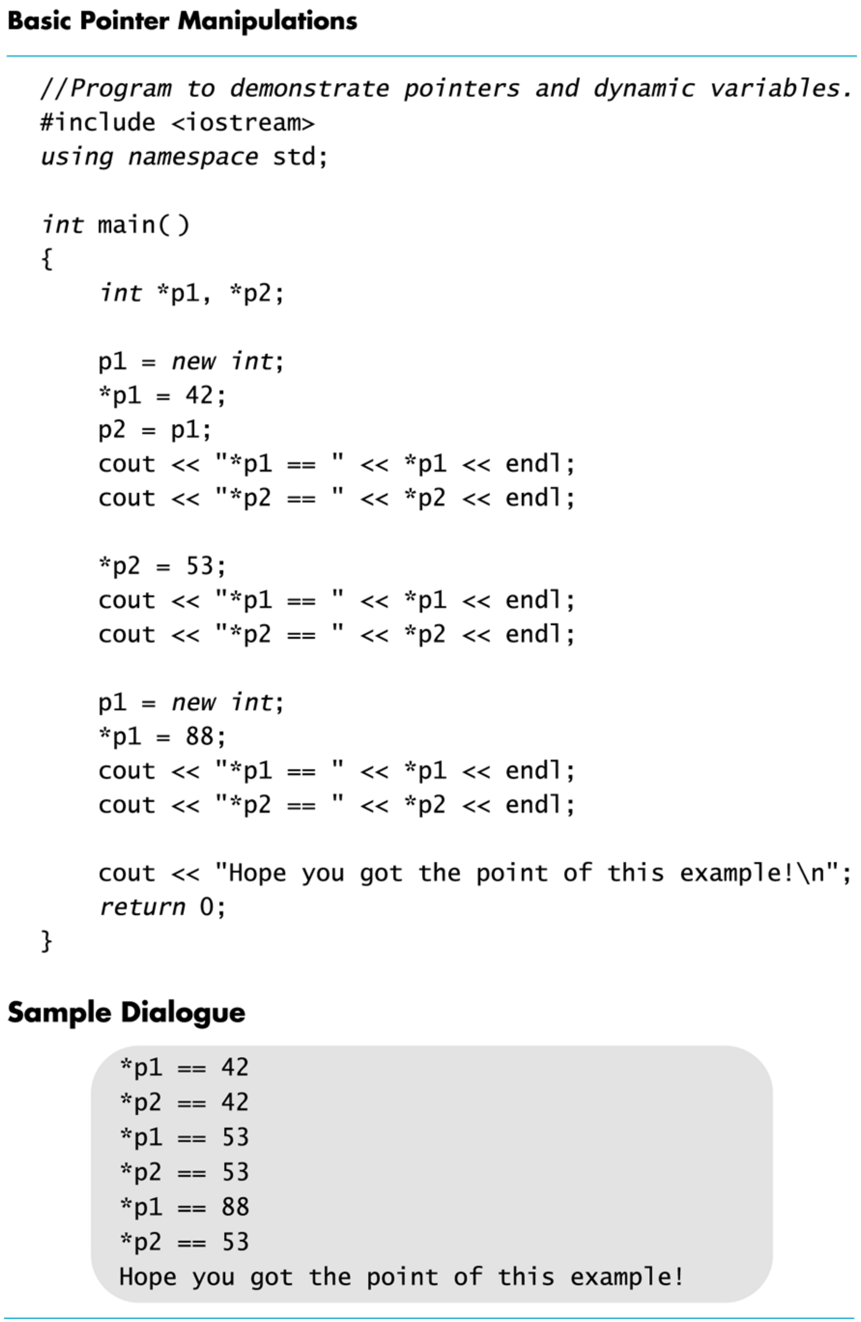
Display 9.1

Uses of the Assignment Operator



Display 9.2

Back Next



Basic Pointer Manipulations

```
//Program to demonstrate pointers and dynamic variables.
#include <iostream>
using namespace std;

int main( )
{
    int *p1, *p2;

    p1 = new int;
    *p1 = 42;
    p2 = p1;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    *p2 = 53;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    p1 = new int;
    *p1 = 88;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    cout << "Hope you got the point of this example!\n";
    return 0;
}
```

Sample Dialogue

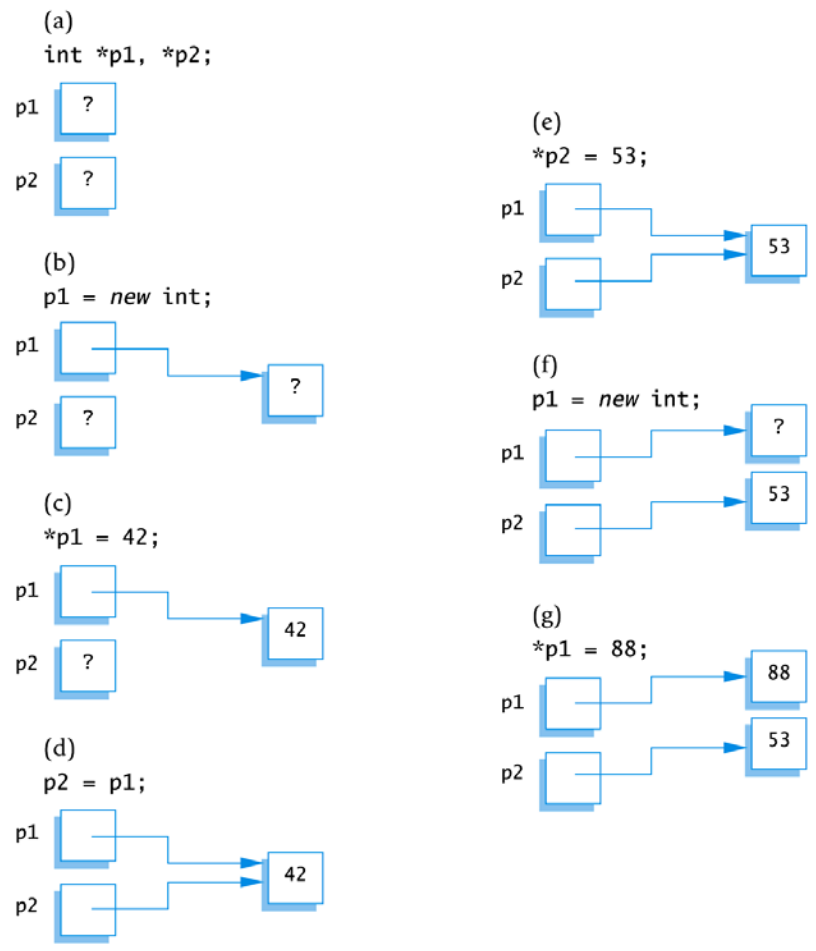
```
*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
```

Display 9.3

Back

Next

DISPLAY 9.3 Explanation of Display 9.2



(a)

```
int *p1, *p2;
```

(b)

```
p1 = new int;
```

(c)

```
*p1 = 42;
```

(d)

```
p2 = p1;
```

(e)

```
*p2 = 53;
```

(f)

```
p1 = new int;
```

(g)

```
*p1 = 88;
```