CSc 10200
Introduction to Computing

Lecture 14
Edgardo Molina
Fall 2013 – City College of New York
References

Syntax:

```cpp
datatype& reference_name = variable;
```

& is the reference operator.
References in action

```cpp
int firstnum = 42;
int secondnum = 23;
int thirdnum = 7;

int& somenum = firstnum;

cout << "the value of somenum is: " << somenum << endl;
firstnum += 10;

cout << "the value of somenum is: " << somenum << endl;
somenum = secondnum;
secondnum -= 10;

cout << "the value of somenum is: " << somenum << endl;
cout << "the value of firstnum is: " << firstnum << endl;
cout << "the value of secondnum is: " << secondnum << endl;
```

Output:
- the value of somenum is: 42
- the value of somenum is: 52
- the value of somenum is: 23
- the value of firstnum is: 23
- the value of secondnum is: 13
References and functions

• A reference variable does not allocate memory
• It only knows the memory address
• References are used:
  – so that memory does not get duplicated
  – to allow functions to return multiple values
  – by default, functions pass a copy of values (PASS BY VALUE)
Returning Multiple Values

• In typical function invocation, called function receives values from its calling function stores and manipulates the passed values, and directly returns at most one value

• **Pass by value:** When data is passed in this manner

• **Pass by reference:** Giving a called function direct access to its calling function’s variables is referred to as

• The called function can reference, or access, the variable whose address has been passed as a pass by reference argument
Passing and Using Reference Parameters

- From the sending side, calling a function and passing an address as an argument that’s accepted as a reference parameter is the same as calling a function and passing a value.
- Whether a value or an address is actually passed depends on the parameter types declared.
Passing and Using Reference Parameters (continued)

Program 6.9

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

void calc(double, double, double, double&, double&); // prototype

int main()
{
    double firstnum, secnum, thirdnum, sum, product;

    cout << "Enter three numbers: ";
    cin >> firstnum >> secnum >> thirdnum;
    calc(firstnum, secnum, thirdnum, sum, product); // function call
    cout << "\nThe sum of the numbers is: " << sum << endl;
    cout << "The product of the numbers is: " << product << endl;

    return 0;
}

void calc(double num1, double num2, double num3, double& total, double& product)
{
    total = num1 + num2 + num3;
    product = num1 * num2 * num3;
    return;
}
```
Passing and Using Reference Parameters (continued)

Figure 6.8 The relationship between argument and parameter names
Passing arrays as arguments
Arrays as Arguments

- An individual array element can be passed as an argument just like any individual variable.
- The called function receives a copy of the array element’s value.
- **Passing an entire array** to a function causes the function to receive a reference to the array, not a copy of its element values.
- The function must be declared with an array as the argument.
- Single element of array is obtained by adding an offset to the array’s starting location.
Arrays as arguments

• Since array is passed as reference, we must also pass the size of the array.

```c
int findmax(int [], int);
```

```c
int findmax(int vals[], int size) {
    int i;
    int max = vals[0];

    for (i=1; i<size; i++) {
        if (max < vals[i]) {
            max = vals[i];
        }
    }

    return max;
}
```

`int []` is a reference to an array.
Arrays as Arguments (continued)

Figure 7.9  The array’s starting address is passed
Arrays as Arguments (continued)

Figure 7.10  Storage of the val array

\[ \text{Offset} = [(3 \times 4) + [1 \times (4 \times 4)]] = 28 \text{ bytes} \]
Internal Array Element Location Algorithm

• Each element of an array is obtained by adding an offset to the starting address of the array:
  – Address of element $i = \text{starting array address} + \text{the offset}$

• Offset for one dimensional arrays:
  – Offset = $i \times \text{the size of the element}$

• Offset for two dimensional arrays:
  – Offset = $\text{column index value} \times \text{the size of an element}$
    $+ \text{row index value} \times \text{number of bytes in a complete row}$
Internal Array Element Location Algorithm (continued)

Figure 7.11  The offset to the element with an index value of 5