Chapter 4
Parameters
Parameters

- Two methods of passing arguments to parameters
- Call-by-value
  - Also called Pass-by-value
  - “copy” of value is passed
- Call-by-reference
  - Also called Pass-by-reference
  - “address of” argument is passed
  - Cannot be used when passing literals
Call-by-Value Parameters

• Copy of actual argument passed
• Considered “local variable” inside function
• If modified, only “local copy” changes
  – Function has no access to “actual argument” from caller
• This is the default method
  – Used in all examples so far
1    //Law office billing program.
2    #include <iostream>
3    using namespace std;
4
4    const double RATE = 150.00;  //Dollars per quarter hour.
5
5    double fee(int hoursWorked, int minutesWorked);
6    //Returns the charges for hoursWorked hours and
7    //minutesWorked minutes of legal services.
8
8    int main()
9    {
10       int hours, minutes;
11       double bill:
12       cout << "Welcome to the law office of\n"
13          << "Dewey, Cheatham, and Howe.\n"
14          << "The law office with a heart.\n"
15          << "Enter the hours and minutes"
16          << " of your consultation:\n"
17       cin >> hours >> minutes;
18       bill = fee(hours, minutes);
19
19       cout.setf(ios::fixed);
20       cout.setf(ios::showpoint);
21       cout.precision(2);
22       cout << "For " << hours << " hours and " << minutes
23           << " minutes, your bill is $" << bill << endl;
24
24       return 0;
25    }

The value of minutes is not changed by the call to fee.
Call-by-Value Example

Display 4.1  Formal Parameter Used as a Local Variable

```java
double fee(int hoursWorked, int minutesWorked) {
    int quarterHours;
    minutesWorked = hoursWorked*60 + minutesWorked;
    quarterHours = minutesWorked/15;
    return (quarterHours*RATE);
}
```

**Sample Dialogue**

Welcome to the law office of Dewey, Cheatham, and Howe. The law office with a heart. Enter the hours and minutes of your consultation:

5 46

For 5 hours and 46 minutes, your bill is $3450.00
Call-by-Reference

• Used to provide access to caller's actual argument

• Caller's data can be modified by called function

• Typically used for input function
  – To retrieve data for caller
  – Data is then “given” to caller

• Specified by ampersand, &, after type in parameter list e.g. int& x;
Call-by-Reference Example

Display 4.2  Call-by-Reference Parameters

```cpp
1 //Program to demonstrate call-by-reference parameters.
2 #include <iostream>
3 using namespace std;
4
4  void getNumbers(int& input1, int& input2);
5  //Reads two integers from the keyboard.
6
6  void swapValues(int& variable1, int& variable2);
7  //Interchanges the values of variable1 and variable2.
8
8  void showResults(int output1, int output2);
9  //Shows the values of variable1 and variable2, in that order.
10 int main()
11 {
12   int firstNum, secondNum;
13
14   getNumbers(firstNum, secondNum);
15   swapValues(firstNum, secondNum);
16   showResults(firstNum, secondNum);
17   return 0;
18 }
```
void getNumbers(int& input1, int& input2)
{
    cout << "Enter two integers: ";
    cin >> input1
    >> input2;
}

void swapValues(int& variable1, int& variable2)
{
    int temp;
    temp = variable1;
    variable1 = variable2;
    variable2 = temp;
}

void showResults(int output1, int output2)
{
    cout << "In reverse order the numbers are: "
    << output1 << " " << output2 << endl;
}

Display 4.2  Call-by-Reference Parameters

SAMPLE DIALOGUE
Enter two integers: 5 6
In reverse order the numbers are: 6 5
Call-by-Reference Details

- What's really passed in?
- A “reference” back to caller's argument
  - Reference to memory location of argument
  - Called “memory address”
  - Unique number referring to a distinct place in memory
Constant Reference Parameters

- Reference arguments can be dangerous
  - Caller's data can be changed
  - Often this is desired, sometimes not

- To protect data & still pass-by-reference
  - Use const keyword
  - No copy is made, only reference is passed
  - Argument is now read-only

```cpp
void displayData(const int& value1, const int& value2);
```
Mixed Parameter Lists

• Can combine passing mechanisms

• Parameter list can include
  – Pass-by-value
  – Pass-by-reference
  – Const pass-by reference

• Order of arguments in function call is critical
  – Pay attention to the order defined in the function header
Parameter Names

• Same rules as naming variables
  – Must be meaningful
  – Should reflect its use

• Functions are self-contained modules
  – Designed separately from rest of program
  – Parameter names can be same as argument names

• Function names should also be meaningful