Parameters

- When a function is executed, it uses the arguments given to it in the function call.
- C++ supports two kinds of parameters
  - Value parameters - the function receives a copy of the argument's value
  - Reference parameters (& after data type) - the function receives the memory address of the caller's argument
- Examples:

```
void Example (int& param1,    // reference
              int  param2,    // value
              float param3 )  // value
```


Value Parameters

- Because value parameters are passed copies of their arguments, anything that has a value may be passed to a value parameter, including constants, variables and complicated expressions.
- Examples

```
PrintLines(3);
PrintLines(lineCount);
PrintLines(2 * abs(10 – someInt));
```

- There must be the same number of arguments in a function call as there are parameters in the function heading. Also, each argument should have the same data type as the parameter in the same position (if not, they are coerced).

Important Note

- When a function returns, the contents of its value parameters are destroyed, along with the contents of its local variables.
- The difference between value parameters and local variables is that the values of local variables are undefined when a function starts to execute, whereas value parameters are automatically initialized to the values of the corresponding arguments.

Reference Parameters

- A function is allowed to inspect and *modify* the value of a reference parameter.
- For a reference parameter, there is only one copy of the information which is used by both the caller and the callee function.
- In essence, the same memory location is known by two different names.

Reference Parameter Example - Activity Problem

- Main
  - Get temperature
  - Print activity
- Get Temperature
  - Prompt user for temperature
  - Read temperature
  - Echo print temperature

Print Activity

```
Print "The recommended activity is"
IF temperature > 85
   Print "swimming."
ELSE IF temperature > 70
   Print "tennis."
ELSE IF temperature > 32
   Print "golf."
ELSE IF temperature > 0
   Print "skiing."
ELSE
   Print "dancing."
```


Activity Program

```
//*************************************************
// Activity program
// This program outputs an appropriate activity
// for a given temperature
//*************************************************

#include <iostream>
using namespace std;

void GetTemp( int& );            // Function prototypes
void PrintActivity( int );
```

Activity Program

```
int main()
{
    int temperature;    // The outside temperature

    GetTemp(temperature);              // Function call
    PrintActivity(temperature);        // Function call
    return 0;
}
```


Activity Program (Continued)

```
void GetTemp( int& temp )        // Reference parameter

// This function prompts for a temperature to be
// entered, reads the input value into temp and
// echo-prints it

{
    cout << "Enter the outside temperature:" << endl;
    cin >> temp;
    cout << "The current temperature is " << temp
         << endl;
}
```


Activity Program (Concluded)

```
void PrintActivity( int temp )       // Value parameter
// Given the value of temp, this function prints a
// message indicating an appropriate activity
{   cout << "The recommended activity is ";
    if (temp > 85)
        cout << "swimming." << endl;
    else if (temp > 70)
        cout << "tennis." << endl;
    else if (temp > 32)
        cout << "golf." << endl;
    else if (temp > 0)
        cout << "skiing." << endl;
    else
        cout << "dancing." << endl;
}
```


Matching Arguments with Parameters

- Only a variable can be passed as an argument to a reference parameter because a function can assign a new value to the argument.
- In contrast, an arbitrarily complicated expression can be passed to a value parameter.
- Consider the following function heading:

```
void DoThis (float val,    //Value
             int& count)   //Reference
```

- Then, the following function calls are valid

```
DoThis(someFloat, someInt)
DoThis(9.83, intCounter)
DoThis(4.9 * sqrt(y), myInt);
```


More About Reference Parameters

- With reference parameters, the argument and the parameter become synonyms for the same memory location, but only during a particular call to the function.
- With value parameters, implicit type coercion occurs if the matched items have different data types. In contrast, with reference parameters, the matched items must have exactly the same type.

Designing Functions

- The specification of what a function does and how it is invoked defines its interface.
- By hiding a module implementation, we can make changes to it without changing the main function, as long as the interface remains the same.
- Designing a function should be divided into two tasks:
  - Designing the interface
  - Designing the implementation

Designing a Function Interface

- Identify the communication that occurs with the function, list
  - Incoming values that the function receives from the caller (value)
  - Outgoing values that the function produces and returns to the caller (reference)
  - Incoming/outgoing values – values the caller has that the function receives and returns (reference)

Function Interface Examples

```
void PrintAverage( /* in */ float sum,
                   /* in */ int count)
// Precondition:
//    sum is assigned && count > 0
// Postcondition:
//    The average sum/count has been output on one line

void Calc( /* in */  int alpha,
           /* inout */ int& beta )
// Precondition:
//    alpha and beta are assigned
// Postcondition:
//    beta == beta@entry * 7 - alpha
```


Problem-Solving Case Study – Comparison of Furniture-Store Sales

- Problem: The regional sales manager wants monthly, department-by-department comparisons, in the form of bar graphs, of two Chippendale stores in town.
- Input: Two data files, each containing the following values for each department:
  - Department ID numer (`int`)
  - Number of business days (`int`)
  - Daily sales values (several `float` values)
- Output: A bar chart showing total sales for each department.

Problem-Solving Case Study (Continued)

- Assumptions: Each file is in order by department ID. Both stores have the same departments.
- Functions:
  - OpenForInput
  - PrintHeading
  - GetData
  - PrintData

Main

Open data files for input
IF either file could not be opened
    Terminate program
Print heading
Get data for a Store 1 department
Get data for a Store 2 department
WHILE NOT EOF on file store1 AND NOT EOF on file store2
    Print data for the Store 1 department
    Print data for the Store 2 department
    Get data for a Store 1 department
    Get data for a Store 2 department

Print Data (In: deptID, storeNum, deptSales)

Print deptID
Print storeNum
WHILE deptSales > 250.0
    Print a '*'
    Subtract 500.00 from deptSales
Terminate current output line

Case Study: Implementation

```
//*******************************************************
// Graph program
// This program generates bar graphs of monthly sales
// by department for two Chippendale furniture stores,
// permitting department-by-department comparison of sales
//*******************************************************

#include <iostream>
#include <iomanip>     // For setw()
#include <fstream>     // For file I/O
#include <string>      // For string type

using namespace std;
```


Case Study: Implementation

```
void GetData( ifstream&, int&, float& );
void OpenForInput( ifstream& );
void PrintData( int, int, float );
void PrintHeading();

int main()
{
    int      deptID1;      // Department ID for Store 1
    int      deptID2;      // Department ID for Store 2
    float    sales1;       // Department sales for Store 1
    float    sales2;       // Department sales for Store 2
    ifstream store1;       // Accounting file for Store 1
    ifstream store2;       // Accounting file for Store 2
```


Case Study: Implementation

```
    cout << "For Store 1," << endl;
    OpenForInput(store1);
    cout << "For Store 2," << endl;
    OpenForInput(store2);

    if ( !store1 || !store2 )        // Make sure files
        return 1;                    //   were opened
    PrintHeading();

    GetData(store1, deptID1, sales1);   // Priming reads
    GetData(store2, deptID2, sales2);
```


Case Study: Implementation

```
   while (store1 && store2)         // While not EOF...
    {
        cout << endl;
        PrintData(deptID1, 1, sales1);  // Process
                                        // Store 1
        PrintData(deptID2, 2, sales2);  // Process
                                        // Store 2
        GetData(store1, deptID1, sales1);
        GetData(store2, deptID2, sales2);
    }
    return 0;
}
```


Case Study: Implementation

```
void OpenForInput( /* inout */ ifstream& someFile )

// Prompts the user for the name of an input file
// and attempts to open the file
// Postcondition:
//     The user has been prompted for a file name
//  && IF the file could not be opened
//         An error message has been printed
// Note:
//     Upon return from this function, the caller must test
//     the stream state to see if the file was successfully
//     opened
```


Case Study: Implementation

```
{
   string fileName;    // User-specified file name

   cout << "Input file name: ";
   cin >> fileName;

   someFile.open(fileName.c_str());

   if ( !someFile )
       cout << "** Can't open " << fileName << " **"
            << endl;
}
```

Case Study: Implementation

```
void PrintHeading()

// Prints the title for the bar chart, a heading, and the
// numeric scale for the chart. The scale uses one mark
// per $500.
// Postcondition:
//     The heading for the bar chart has been printed

{
   cout << "Bar Graph Comparing Departments of Store #1
            and Store #2" << endl << endl << "Store
            Sales in 1,000s of dollars" << endl
    << "#0         5        10        15        20        25"
    << endl
    << " |.........|.........|.........|.........|.........|"
    << endl;
}
```


Case Study: Implementation

```
void GetData( /* inout */ ifstream& dataFile,
              /* out */   int&      deptID,
              /* out */   float&    deptSales )

// Takes an input accounting file as a parameter, reads
// the department ID number and number of days of sales
// from that file, then reads one sales figure for each
// of those days, computing a total sales figure for
// the month.  This figure is returned in deptSales.
// (If input of the department ID fails due to
// end-of-file, deptID and deptSales are undefined.)
// Precondition:
//     dataFile has been successfully opened
//  && For each department, the file contains a
//     department ID, number of days, and one sales
//     figure for each day
```


Case Study: Implementation

```
// Postcondition:
//     IF input of deptID failed due to end-of-file
//          deptID and deptSales are undefined
//     ELSE
//          The data file reading marker has advanced
//          past one department's data
//       && deptID == department ID number as read from
//          the file
//       && deptSales == sum of the sales values for
//          the department

{
    int   numDays;  // Number of business days in the month
    int   day;      // Loop control variable for daily sales
    float sale;     // One day's sales for the department
```


Case Study: Implementation

```
    dataFile >> deptID;
    if ( !dataFile )         // Check for EOF
        return 1;            // If so, exit the function

    dataFile >> numDays;
    deptSales = 0.0;
    day = 1;          // Initialize loop control variable
    while (day <= numDays)
    {
        dataFile >> sale;
        deptSales = deptSales + sale;
        day++;            // Update loop control variable
    }
}
```


Case Study: Implementation

```
void PrintData( /* in */ int   deptID,
                /* in */ int   storeNum,
                /* in */ float deptSales )

// Prints the department ID number, the store number, and
// a bar graph of the sales for the department. The bar
// graph is printed at a scale of one mark per $500.
// Precondition:
//     deptID contains a valid department number
//  && storeNum contains a valid store number
//  && 0.0 <= deptSales <= 25000.0
// Postcondition:
//     A line of the bar chart has been printed with one
//     * for each $500 in sales, with remainders over
//     $250 rounded up
//  && No stars have been printed for sales <= $250
```


Case Study Implementation (Concluded)

```
{
    cout << setw(12) << "Dept " << deptID << endl;
    cout << setw(3) << storeNum << "     ";

    while (deptSales > 250.0)
    {
        cout << '*' ;        // Print '*' for each $500

        // Update loop control variable
        deptSales = deptSales - 500.0;

    }
    cout << endl;
}
```