Using Reference Variables as Parameters

- A mechanism that allows a function to work with the **original argument** from the function call, not a copy of the argument
- Allows the function to **modify values** stored in the calling environment
- Provides a way for the function to 'return' more than one value

Passing by Reference

- A reference variable is an alias for another variable
- Defined with an ampersand (`&`)

```
void getDimensions(int&, int&);
```

- Changes to a reference variable are made to the variable it refers to
- Use reference variables to implement passing parameters *by reference*

Passing by Reference – Example

The & here in the prototype indicates that the parameter is a reference variable.

Program 6-24

```
1  // This program uses a reference variable as a function
2  // parameter.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype. The parameter is a reference variable.
7  void doubleNum(int &);
8
9  int main()
10 {
11    int value = 4;
12
13    cout << "In main, value is " << value << endl;
14    cout << "Now calling doubleNum..." << endl;
15    doubleNum(value);
16    cout << "Now back in main. value is " << value << endl;
17    return 0;
18 }
19
```

Here we are passing value by reference.

Passing by Reference - Example

Program 6-24 *(Continued)*

The & also appears here in the function header.

```
20  //*************************************************
21  // Definition of doubleNum.                       *
22  // The parameter refVar is a reference variable. The value *
23  // in refVar is doubled.                          *
24  //*************************************************
25
26  void doubleNum (int &refVar)
27  {
28     refVar *= 2;
29  }
```

Program Output

```
In main, value is 4
Now calling doubleNum...
Now back in main. value is 8
```

Reference Variables Notes

- Each reference parameter must contain &
- Space between type and & is unimportant
- Must use & in both prototype and header
- Argument passed to reference parameter must be a variable – cannot be an expression or constant
- Use when appropriate – don't use when argument should not be changed by function, or if function needs to return only 1 value

```
#include <iostream>
using namespace std;
void test(int, int&);

int main()
{
    int num;
    num=5;
    test(24, num);
    cout<<num<<endl;
    test(num,num);
    cout<<num<<endl;
    test(num*num, num);
    cout<<num<<endl;
    test(num+num,num);
    cout<<num<<endl;
    system("pause");
    return 0;
}
```

```
void test(int first, int& second)
{
    int third;
    third=first+second*second+2;
    first=second-first;
    second=2*second;
    cout<<first<<" "<<second<<"
  "<<third<<endl;
}
```

```
#include <iostream>
using namespace std;
void test(int&, int&,int,int&);

int main()
{    int a,b,c,d;
   a=3;  b=4;  c=20;  d=78;
 cout<<a<<" "<<b<<" "<<c<<"
   "<<d<<endl;
    test(a,b,c,d);
 cout<<a<<" "<<b<<" "<<c<<"
   "<<d<<endl;
    d=a*b+c-d;
    test(a,b,c,d);
    cout<<a<<" "<<b<<" "<<c<<"
   "<<d<<endl;
    return 0;
}
```

```
void test(int& a, int& b, int c, int&
   d)
{
      cin>>a >> b>> c>> d;
      c = a* b+d-c;
      c=2*c;
}
```

The input:

6 8 12 35
8 9 30 45

Default Arguments

- A Default argument is an argument that is passed automatically to a parameter if the argument is missing on the function call.
- Must be a constant declared in prototype:

```
void evenOrOdd(int = 0);
```

- Can be declared in header if no prototype
- Multi-parameter functions may have default arguments for some or all of them:

```
int getSum(int, int=0, int=0);
```

Default Arguments – Example

Default arguments specified in the prototype

Program 6-23

```
1  // This program demonstrates default function arguments.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype with default arguments
6  void displayStars(int = 10, int = 1);
7
8  int main()
9  {
10    displayStars();      // Use default values for cols and rows.
11    cout << endl;
12    displayStars(5);     // Use default value for rows.
13    cout << endl;
14    displayStars(7, 3);  // Use 7 for cols and 3 for rows.
15    return 0;
16 }
```

(Program Continues)

Default Arguments – Example

Program 6-23 *(Continued)*

```
18  //*********************************************************
19  // Definition of function displayStars.                   *
20  // The default argument for cols is 10 and for rows is 1.*
21  // This function displays a square made of asterisks.     *
22  //*********************************************************
23
24  void displayStars(int cols, int rows)
25  {
26     // Nested loop. The outer loop controls the rows
27     // and the inner loop controls the columns.
28     for (int down = 0; down < rows; down++)
29     {
30        for (int across = 0; across < cols; across++)
31           cout << "*";
32        cout << endl;
33     }
34  }
```

Program Output

```
**********

*****

*******
*******
*******
```

Default Arguments

- If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list:

```
int getSum(int, int=0, int=0);// OK
int getSum(int, int=0, int);  // NO
```

- When an argument is omitted from a function call, all arguments after it must also be omitted:

```
sum = getSum(num1, num2);     // OK
sum = getSum(num1, , num3);   // NO
```

Default Arguments

- Consider the following function prototype:

```
void funcExp(int,int, double=55.5,char='A');
```

- The following function calls are legal:
  - `funcExp(3,4,45.5, 'B');`
  - `funcExp(3,4,45.5);`
  - `funcExp(3,4);`
- The following function calls are illegal:
  - `funcExp(3,4, 'C');`

Default Arguments

- The following are illegal function prototypes with default arguments:
  - `void funcOne(int,double=23.45,char, int=45);`
  - `int funcTwo(int=1, int, int=1);`
  - `void funcThree(int, int&=16,double=34);`

- Consider the following function prototype & function definition:

```
void testDefaultParam(int , int=5, double=3.2);


void testDefaultParam(int a, int b, double z)
{
    int u;
    a=a+static_cast<int>(2*b+z);
    u=a+b*z;
    cout<<"u = "<<a<<endl;
}
```

What is the output of the following function calls?

a) testDefaultParam(6);

b) testDefaultParam(3,4);

c) testDefaultParam(3,4.5);

d) testDefaultParam(3,4, 5.5);

e) testDefaultParam(3.4);

In-Class Exercise

- Write a function prototype and function header for a function called `compute`. The function should have 3 parameters: an `int`, a `double` and a `long`. The `int` parameter should have a default argument of 5, and the `long` parameter should have a default argument of 65536. The `double` parameter should have no default arguments. The parameters no necessarily in the order.

- Write a function prototype and function header for a function called `calculate`. The function should have 3 parameters: an `int`, a reference to a `double` and a `long`. Only the `int` parameter should have a default argument, which is 47. The parameters no necessarily in the order.

Overloading Functions

- Overloaded functions have the same name but different parameter lists
- Can be used to create functions that perform the same task but take **different parameter types or different number of parameters**
- Compiler will determine which version of function to call by argument and parameter lists

Function Overloading Examples

- Using these overloaded functions,

```
void getDimensions(int);              // 1
void getDimensions(int, int);         // 2
void getDimensions(int, double);      // 3
void getDimensions(double, double);// 4
```

- the compiler will use them as follows:

```
int length, width;
double base, height;
getDimensions(length);              // 1
getDimensions(length, width);       // 2
getDimensions(length, height);      // 3
getDimensions(height, base);        // 4
```

Function Overloading – Example

Program 6-26

```
1  // This program uses overloaded functions.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  // Function prototypes
7  int square(int);
8  double square(double);
9
10 int main()
11 {
12    int userInt;
13    double userFloat;
14
15    // Get an int and a double.
16    cout << fixed << showpoint << setprecision(2);
17    cout << "Enter an integer and a floating-point value: ";
18    cin >> userInt >> userFloat;
19
20    // Display their squares.
21    cout << "Here are their squares: ";
22    cout << square(userInt) << " and " << square(userFloat);
23    return 0;
24 }
```

The overloaded functions have different parameter lists

Passing a double

Passing an int

(Program Continues)

Function Overloading – Example

Program 6-26 *(Continued)*

```
26  //*************************************************************
27  // Definition of overloaded function square.                  *
28  // This function uses an int parameter, number. It returns the *
29  // square of number as an int.                                *
30  //*************************************************************
31
32  int square(int number)
33  {
34     return number * number;
35  }
36
37  //*************************************************************
38  // Definition of overloaded function square.                  *
39  // This function uses a double parameter, number. It returns  *
40  // the square of number as a double.                          *
41  //*************************************************************
42
43  double square(double number)
44  {
45     return number * number;
46  }
```

Program Output with Example Input Shown in Bold

```
Enter an integer and a floating-point value: 12 4.2 [Enter]
Here are their squares: 144 and 17.64
```

The `exit()` Function

- **Terminates** the execution of a program
- Can be called from any function
- Can pass an `int` value to operating system to indicate status of program termination
- Usually used for **abnormal termination** of program
- Requires `cstdlib` header file (Borland)

The `exit()` Function

- Example:

```
exit(0);
```

- The `cstdlib` header defines two constants that are commonly passed, to indicate success or failure:

```
exit(EXIT_SUCCESS);
exit(EXIT_FAILURE);
```

In-Class Exercise

- What is the output for the following programs

```
#include <iostream>
using namespace std;
void function();
int main(){
   function();
   cout << "Bye from main.\n";
   system ("pause");   return 0;
}

void function(){
   cout << "Bye! from function
before exit\n";
   exit(0);
   cout << "Bye! from function
before exit\n";
}
```

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

int function();
int main(){
   function();
   cout << "Bye from main.\n";
   system ("pause"); return 0;
}

int function(){
   cout << "Bye! from function
before return\n";
   return 0;
   cout << "Bye! from function
before return\n";
}
```

In-Class Exercise

- Write a program that calculates the average of a group of test scores, where the lowest score in the group is dropped. It should use the following functions:
  - `getScore` – This function ask the user for a test score, store it in a reference parameter variable, and validate it. For input validation, do not accept test scores lower than 0 or higher than 100. This function should be called by `main()` once for each of the five scores to be entered.
  - `calcAverage` – This function calculates and display the average of the four highest score. This function should be called just once by `main()`, by should be passed the five scores.
  - `findLowest` – This function finds and returns the lowest of the five scores passed to it. It should be called by `calcAverage` function, which uses the function to determine which of the five scores to drop.