1 Parameter Passing Techniques

- Pass by Value
- Pass by Reference
- Pass by Pointer

1.1 Parameter Passing—Pass by Reference

- Don't want to pass by value always.
- Used when we want to *return* more than one value from a function.

1.2 Swapping two values

```
int main()
{
    int  n1, n2;

    cout << "Enter two numbers: " << flush;
    cin >> n1 >> n2;
    if( n1 > n2 )
       Swap( n1, n2 );

    cout << "Sorted order: " << n1 << ", " << n2 << endl;

    return 0;
}
```

How to write `swap()`?

```
    //  WARNING:  This swaps the values inside the function
void Swap( int n1, int n2 )
{
    int temp = n1;
    n1 = n2;
    n2 = temp;
}
```

This won't work! If we change from pass by value (default parameter passing mechanism) to pass by reference, it will. Note the changes made to `Swap()`!

```
void Swap( int& n1, int& n2 )
{
    int temp = n1;
    n1 = n2;
    n2 = temp;
}
```

Another way to write this function is by using pointers (more about that later). To use this, we also need to change our call to `Swap()`—`Swap( &n1, &n2 );`

```
void Swap( int* n1, int* n2 )
{
    int temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Note how strange this looks—especially since we haven't discussed pointers! Passing by reference is much cleaner than using pointers and it accomplishes the same general function.

1.3 Basic Pointer Concepts

- Point to a memory location.
- Call by reference is based on pointers.
- Operators:
  - `&` Address operator
  - `*` Dereferencing operator
- Machine/compiler dependencies exist.
- Care and caution should be exercised when using pointers.

Pointers will be extensively in later Computer Science courses—unless everything moves to Java. There are no pointers in Java.

1.4 Pointer examples

```
int  a;
int  *aPtr;

a = 5;
aPtr = &a;
cout << a << endl;
cout << *aPtr << endl;    //  contents of a
*aPtr = 6;
cout << a << endl;
cout << *aPtr << endl;    //  contents of a
cout << &a    << endl;    //  address of a (compiler/machine dependent)
```

Output:

```
5
5
6
6
0x024b2fa8
```

1.5 Example: reading values from a file

A function that retrieves three values from an input file stream.
Verifies that the x-value is greater than zero and that the y-value is less than ten.

```
int GetData( ifstream& inF, double &x, double& y, double& z )
{
    int rCode = 1;  //  return code

    inF >> x >> y >> z;

    if( x < 0.0 )
    {
       cerr << "Unexpected negative value for x" << endl;
       rCode = -1;
    }

    if( y > 10.0 )
    {
       cerr << "Unexpected value for y" << endl;
       rCode = -1;
    }

    return rCode;
}
```

Why is this good? Checks data integrity and localizes error messages. What if we want to change our validity checking? Not so good—a general purpose function(s) would be useful.

```
//  quadParams.cpp

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>

    //  prototypes
void ReadCoefficients( double& a, double& b, double& c );
void CalculateRoots( double a, double b, double c );
void OpenInputAndOutputFiles();

    //  global variables
ifstream fIn;
ofstream fOut;

int main()
{
    OpenInputAndOutputFiles();

    char  ans;           //  answer
    double  a, b, c;     //  coefficients

    do
    {
            //  Get coefficients from file
        ReadCoefficients( a, b, c );

        CalculateRoots( a, b, c );

        do
        {
                //  prompt
            fOut << "Continue (y/n)? " << flush;
                //  read answer
            fIn >> ans;

            fOut << ans << endl;
```

```
        } while ( (ans != 'y') && (ans != 'n') );
    } while ( (ans == 'y') );

    cout << " Done!" << endl;

    return  0;
}
```

```
void ReadCoefficients( double& a, double& b, double& c )
{
    fIn >> a >> b >> c;

        //  Echo values
    fOut << "\nThe polynomial coefficients are: " << endl;
    fOut << "a:  " << a << endl;
    fOut << "b:  " << b << endl;
    fOut << "c:  " << c << endl;
}
```

```
void CalculateRoots( double a, double b, double c )
{
    double discr = b*b - 4.0*a*c;

    if( discr <= 0.0 )
    {
        fOut << "Unable to solve quadratic equation:" << endl;
        fOut << "\tDiscriminant is less than or equal zero" << endl;
    }
    else
    {
        discr = sqrt(discr);
        double denom = 2.0 * a;
        double x1 = (-b + discr) / denom;
        double x2 = (-b - discr) / denom;

        fOut << "x1: " << x1 << endl;
        fOut << "x2: " << x2 << endl;
    }
}
```

```
void OpenInputAndOutputFiles()
{
        //  Open input file
    fIn.open( "quad.in", ios::in );
    if( !fIn )              //  verify file was opened
    {
        cerr << "Unable to open input file:  quad.in" << endl;
        exit( -1 );
    }

    //  Open output file
    fOut.open( "quad.out", ios::out );
    if( !fOut )              //  verify file was opened
    {
        cerr << "Unable to open output file:  quad.out" << endl;
        exit( -1 );
    }
}
```

Output:

```
The polynomial coefficients are:
a:  2
b:  3
c:  5
Unable to solve quadratic equation:
    Discriminant is less than or equal zero
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  5
c:  3
x1: -1
x2: -1.5
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  6
c:  2
x1: -0.381966
x2: -2.61803
Continue (y/n)? n
 Done!
```

2 Function Overloading

What if we want to find the minimum of some number of integers or real numbers? We could write one function and *cast* the arguments to the necessary type (e.g., `double` to `int`). It works, but might lose values and it isn't very neat. Or we can write two functions that use the desired arguments.

```
int Min( int a, int b, int c )
{
}


double Min( double x, double y, double z )
{
}
```

These functions can be used as follows:

```
int main()
{
    int i;
    double x;

    i = Min( 3, 1, 5 );
    x = Min( 2.71, 1.01, 5.23 );
}
```

The compiler resolves which function to call.

- If a function defintion exists where the type of the parameters exactly match, that function is used.
- If there is not an exact match, the compiler will *cast* the parameters.
- Rules are complicated.
- Be careful when using function overloading.

Function overloading is very useful when doing Object-Oriented Programming, especially when creating objects.