Lab 5 08227 Advanced Programming

This tutorial introduces the reader to parameter passing

1.0 Setup

Download the Lab5.zip file and extract the files to G:/08227/Lab5/.

Create a new empty C++ project.

Add the **source.cpp** to the new project.

The **source.cpp** file contains the following code:

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

void swap(int lhs, int rhs)
{
     int temp = lhs;
     lhs = rhs;
     rhs = temp;
}

void main(int, char**) {

     int a = 10;
     int b = 20;

     cout << "a=" << a << ", b=" << b << endl;

     swap(a, b);

     cout << "a=" << a << ", b=" << b << endl;

     system("PAUSE");
}
```

We want our program to swap the values of the two variables **a** and **b**.

Compile and run the program.

It does not give the correct answer; **a** and **b** are not swapped.

2.0 Pass by value

View **source.cpp** within Visual Studio.

Place a breakpoint on:

```
swap(a, b);
```

Run the program. Execution should stop at the breakpoint. Now open both the dissembler window and the register window.

Open a memory window and set it to look at the location of the stack (see previous lab)

You are now in a position to debug the code.

Single step through the assembly, watching how the values of **a** and **b** are pushed onto the stack.

Remember to use F11 and not F10, so you can jump into the **swap** function

When you get to the following line, F11 will not appear to do anything. This assembly instruction is actually a very small loop, consisting of one line of code. Repeated presses of F11 will execute each iteration of the loop in turn. F10 will execute the entire loop and jump to the next instruction.

```
00165D6C  rep stos    dword ptr es:[edi]
```

Remember to keep an eye on the Locals window to see the values of your C++ variables.

At the end of the swap function the values of the variables have been swapped. You can see in the Locals window.

When you return from the swap function, notice how no data is actually copied back to the original variables.

3.0 Pass by reference

Rewrite the previous code to pass the variables **a** and **b** into the swap function by reference, rather than by value.

Compile and run you code. Did this swap the values? It should have worked.

Repeat the debugging process from the previous section. Place a breakpoint on the **swap** call, run the program and then disassemble

Single step through the calling process.

Notice how the values of 10 and 20 are no longer pushed onto the stack. Instead the address of the variables **a** and **b** are pushed onto the stack.

Find the address of **a** in the appropriate register, and use the address to look at the memory that holds the value for the variable **a**.

4.0 Pass by address (optional)

Repeat the previous section but this time rewrite the swap function to pass the parameters by address.

5.0 Return by value

Replace the swap function with the **clamp** function

```
int clamp(int value, int low, int high)
{
     if (value < low)
        return low;
     if (value > high)
        return high;
     return value;
}
```

This function clamps or limits a value between an upper and lower bound.

Add a call to the **clamp** function in **main**.

Repeat the dissembling process but this time focus on the return value.

Notice how the return value is copied onto the stack.

Check the value is correct on the stack.

6.0 Return by reference

Add this alternative **clamp** function

```
int& clamp(int& value, int low, int high)
{
     if (value < low)
        return low;
     if (value > high)
        return high;
     return value;
}
```

Repeat the dissembling process focusing on the return value.

Notice how the return value is copied onto the stack, but unlike the previous exercise this return value is actually the address of the variable **value**

Check that this address is correctly stored on the stack.