Passing arguments/parameters in C++ is a very common occurance and is necessary in many programs or functions within programs. Passing parameters is a necessity and when executed in C++ the code is simple and often used. Since C++ is a high level language that executes more machine oriented code that is not seen or looked at very often. Looking at the assembly code creates more of an appreciation of the implementation that goes on "behind the scenes" of the C++ execution.

There are a couple of ways to pass parameters in C++: by value, by reference and by pointer. Each of these provides its own benefits and serves its own purpose. Passing by value is exactly what it sounds like, you literally put a value in as a parameter. When this is done in assembly, the code that is generally used is:

```
mov    EAX, DWORD PTR [ESP + 16]
mov    ECX, DWORD PTR [ESP + 12]
```

This segment of assembly code is declaring the space for the required variables in the method and it will use EAX and ECX to pass these values into the method itself. The C++ code that generated this NASM code is a simple xToN function that takes in two int values and computes the value of x raised to the n power. When calling this method in main, the assembly code assigns the values to the EAX and the ECX register (x and n or 8 and 2 in this case, respectively) and then adjusts the pointer values and assigns the given values:

```
mov    EAX, 8
mov    ECX, 2
mov    DWORD PTR [EBP - 4], 0
mov    DWORD PTR [ESP], 8
mov    DWORD PTR [ESP + 4], 2
mov    DWORD PTR [EBP - 8], EAX # 4-byte Spill
mov    DWORD PTR [EBP - 12], ECX # 4-byte Spill
```

When passing other types of values, the code remains fairly similar in the main method, storing the ascii values for chars, or storing larger words for doubles and floats. In the definition of the function itself, it still uses EAX and ECX but only the smaller portion of it such as: AL and CL.

Another common way of passing parameters is through reference. This, unlike when passing by value, can result in the method modifying the actual parameters or useful for passing larger data structures. When information is passed through reference, it is done by passing the actual address of the information instead of the value. This allows the subroutine to directly access the data itself and change it if necessary. Passing through reference at the assembly level is the same as passing by pointers as references are similar to aliases for pointers.

```
sub    ESP, 24
lea    EAX, DWORD PTR [EBP - 8]
mov    DWORD PTR [EBP - 4], 0
mov    DWORD PTR [EBP - 8], 3
mov    DWORD PTR [ESP], EAX
```

Passing by references leads to the computer allocating memory in the registers differently. For references, register addresses are decremented to accommodate for the larger memory values since references are passed as addresses instead of values. The syntax itself does not change that much but the overall memory allocation changes greatly. Passing parameters also involves passing objects such as arrays into functions and then having both the caller and callee access these arrays to perform the method. Arrays are stored in the registers similar to variables in the method itself, but when arrays are created in main, there is more space allocated to them and are stored like:

```
sub    ESP, 56
lea    EAX, DWORD PTR [.L_ZZ4mainE1x]
mov    ECX, 20
```

```
lea     EDX, DWORD PTR [EBP - 24]
mov     DWORD PTR [EBP - 4], 0
mov     DWORD PTR [ESP], EDX
mov     DWORD PTR [ESP + 4], EAX
mov     DWORD PTR [ESP + 8], 20
mov     DWORD PTR [EBP - 28], ECX # 4-byte Spill
call    memcpy
```

The assembly code calls a function called memcpy to store the additional elements of the array. The memcopy function is as follows (from: http://pastebin.com/wEx3F5hL):

```
mov     eax,[ esp + 12 ]
mov     ebx,[ esp + 4 ]
mov     ecx,[ esp + 8 ]

mem_loop:
        dec     eax
        mov     al,[ ecx + eax ]
        mov     [ ebx + eax ],al
        cmp     eax,0
        jne     mem_loop

    ret     12
```

The arrays are accessed through the EAX register and it is loaded through the ESP (stack pointer). The values given from the gdb state that the array is stored 0xbffff330.