//Yujian Li (yl7kd) 03/29/16 Section 102

InLab08

In this lab we look into a different aspect of computer science: The Assembly Code. During the inlab process, we use the assembly code generated by a test C++ program in order to understand how data was passed in the system.

Parameter passing:

Because there are different types of data, storing in the assembly code will treat each code. When integer was passed by value, it was directly used “mov” command to store value into the designated register “exa” and then call the function.

```
“mov DWORD PTR [esp+28], 1
 mov eax, DWORD PTR [esp+28]
 mov DWORD PTR [esp], eax”
```

While it was passed by reference, it used “lea” command and also store the actual address into the “exa” register.

```
“mov DWORD PTR [esp+28], 1
 lea eax, [esp+28]
 mov DWORD PTR [esp], eax”
```

In the callee section there is also an extra line of “mov eax, DWORD PTR [eax]” comparing to pass by value. When char type was passed by value and reference the assembly code does the similar operation except it uses “BYTE PTR” instead of “DWORD PTR” when storing the values due to the size of the char is less than int. Since passing pointers is already a reference of certain value therefore there is no case of passing by reference for pointers. It uses “movsx” operation inside the callee to store value. When passing by value,

```
“mov   eax, DWORD PTR [esp+28] ,
 mov   DWORD PTR [eax],
 mov   eax, DWORD PTR [esp+28],
 mov   DWORD PTR [esp], eax”
```

were the operations it performed, though it did not use the “lea” operation, it did do the similar operation as the process of passing by reference for int or char. Float operation is also similar with int, especially within the callee section. In the main section, it used “mov eax, DWORD PTR .LC0” to deal with floating numbers. Because there are multiple elements within the object, the elements are stored orderly in the main section with 4 byte away as the pointer address has shown. Because of the stack structure nature, when storing into the stack, the callee will move the last elements into the stack first. For array in assembly code. It works very similar to the object. As the main code has shown, it was passed by reference and inside the callee, the callee will first access its base address then try to access each elements inside array by the operation “add 4”. As we have studied before, the process of passing by reference is very similar to passing by pointers, and also shown in the code int (reference and pointer).

Object:

At the beginning, it initializes all the values inside the object.

```
“mov   DWORD PTR [esp+44], 1
 mov   BYTE PTR [esp+48], 97
 mov   eax, DWORD PTR .LC0
 mov   DWORD PTR [esp+56], eax”
```

It shall then first reserve certain address space on esp for the future use, then it will pass the initialized into the rest esp address space left. (ex: esp+0 to esp+32 is reserved. Int value is stored in esp+36).

Then it will gradually move all the values into eax first then into esp register by 4 bytes. As I saw from the code, private values are stored in the register esp prior to esp+8 while public fields are stored after esp+8 and the callee automatically forms a stack that stores When accessing data members inside array, each values has been previously assigned with a specific register, so the callee will then moves from the base address to access the needed data member. Because the base address of the object has been stored into the register in main(), it will shall just call move the designated register value to eax and call the method related to it. As mentioned before, the nature of the object's address can be passed in as an argument and the function's first argument can also be a object passed by reference, there is no specific difference between invoking methods or invoking functions. The main difference between accessing a private field and a public field is that the private field requires a getter to be accessed which involves the nested call of functions.

```
“lea    eax, [ebp+8]
mov     DWORD PTR [esp], eax”
```

The “lea” operation also shows that it has store this private value into the register instead of the “mov” method for other public typies.

When using “this” pointer command, it serves as a copy of the original class. For example, In my getj() method:

```
“mov   eax, DWORD PTR [ebp+8]
mov    eax, DWORD PTR [eax+16]”
```

it first stores the address of the object, later it requires to increment the array address in order to access the designated element. By updating the base address of the object, the entire get() method shall update its result.

Reference:

https://en.wikibooks.org/wiki/X86_Disassembly/Objects_and_Classes

http://stackoverflow.com/questions/19850437/general-questions-on-c-assembly-data-layout-data-member-access-methods