#Amber Liu

#Al7bf

#11/09/2018

#postlab8.pdf

1. Passing ints by value as a parameter in the image below shows that the program first pushes rbp to continue storing value in rsp. Then the program would mov/copy value in the first parameter to [rbp-x], where x is the size in bytes of the passing parameter, to plant integer into subroutine. The register changes to correspond to the memory it needs for the parameter. The local variable means that the value is stored below rbp which stores rsp. Passing by reference in int, char, pointer, and float are all similar where it locates as a local parameter below rsp. The -8 shows that memory usage used to store parameter is 8 bytes. Below are some examples of int, chars, pointers, floats passed by value and passed by reference.

Passing by value

Int

```
passByVal(int, int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     DWORD PTR [rbp-8], esi
        mov     edx, DWORD PTR [rbp-4]
        mov     eax, DWORD PTR [rbp-8]
        add     eax, edx
        pop     rbp
        ret
```

Char

```
passByVal(char, char):
        push    rbp
        mov     rbp, rsp
        mov     eax, edi
        mov     edx, esi
        mov     BYTE PTR [rbp-4], al
        mov     eax, edx
        mov     BYTE PTR [rbp-8], al
        movsx   edx, BYTE PTR [rbp-4]
        movsx   eax, BYTE PTR [rbp-8]
        add     eax, edx
        pop     rbp
        ret
```

Pointer

```
passPointer(int*):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     DWORD PTR [rax], 20
        nop
        pop     rbp
        ret
```

Float

```
passByVal(float, float):
        push    rbp
        mov     rbp, rsp
        movss   DWORD PTR [rbp-4], xmm0
        movss   DWORD PTR [rbp-8], xmm1
        movss   xmm0, DWORD PTR [rbp-4]
        addss   xmm0, DWORD PTR [rbp-8]
        cvttss2si       eax, xmm0
        pop     rbp
        ret
```

Passing by reference

&int

```
passByRef(int&, int&):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     QWORD PTR [rbp-16], rsi
        mov     rax, QWORD PTR [rbp-8]
        mov     edx, DWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-16]
        mov     eax, DWORD PTR [rax]
        imul    eax, edx
        pop     rbp
        ret
```

&float

```
passByRef(float&, float&):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     QWORD PTR [rbp-16], rsi
        mov     rax, QWORD PTR [rbp-8]
        movss   xmm1, DWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-16]
        movss   xmm0, DWORD PTR [rax]
        mulss   xmm0, xmm1
        cvttss2si       eax, xmm0
        pop     rbp
        ret
```

&char

```
passByRef(char&, char&):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     QWORD PTR [rbp-16], rsi
        mov     rax, QWORD PTR [rbp-8]
        movzx   eax, BYTE PTR [rax]
        movsx   edx, al
        mov     rax, QWORD PTR [rbp-16]
        movzx   eax, BYTE PTR [rax]
        movsx   eax, al
        imul    eax, edx
        pop     rbp
        ret
```

&pointer

```
passByRef(int*&, int*&):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     QWORD PTR [rbp-16], rsi
        mov     rax, QWORD PTR [rbp-16]
        mov     rax, QWORD PTR [rax]
        mov     eax, DWORD PTR [rax]
        pop     rbp
        ret
```

2. Pass by Object

Value

```
show(abc):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        lea     rax, [rbp-4]
        mov     rdi, rax
        call    abc::display()
        mov     esi, eax
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >:
        nop
        leave
        ret
```

Reference

```
show(abc&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    abc::display()
        mov     esi, eax
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >:
        nop
        leave
        ret
```

Assembly does not really have h files like C++, but the values are still kept in memory. First parameter will be put in the address that rdi registers. No matter what the size of field is, next parameter is storing by incrementing pointer by 8 and 16 for the next one.

3. For the array, my program looped through the array and multiplied every element in the array by int x. When the callee is called, parameters were accessed first and rbp pointer begins at the first element. When c++ code increments in the loop, the index in assembly is also increments by rax*4. It is multiplied by 4 because value in the array is an int, where next number is in the next 4 bytes. Since the address of every value is coded in pointer, also had to dereference to access it.
4. The assembly code looks very similar(see photos from before) with the parameter passing and dereference. There isn't any difference in assembly code, where the parameter to store value is the same as the pointer. The memory address is stored as a local variable right below rsp.

Objects

```
student::student(string n, int i, char s){
    name = "im tired";
    i = 0;
    s = 'a';

}
```

```
student::student(std::__cxx11::basic_string<char, std::char_trait
        push    rbp
        mov     rbp, rsp
        push    rbx
        sub     rsp, 40
        mov     QWORD PTR [rbp-24], rdi
        mov     QWORD PTR [rbp-32], rsi
        mov     DWORD PTR [rbp-36], edx
        mov     eax, ecx
        mov     BYTE PTR [rbp-40], al
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits
        mov     rax, QWORD PTR [rbp-24]
        mov     esi, OFFSET FLAT:.LC0
        mov     rdi, rax
```

1. Assembly is able to store different types together in one class such that one class acts as base pointer where other field has its specific “pointer” by changing the base pointer to define its position when called. The pointer is like a padding of data alignment such that it performs in a way like array storing the different types.
2. The program access member via member function, the program uses callee to store the local variable. Private data members however, cannot be accessed when calling outside in C++. But in assembly code, it doesn't differ between private or public, so program can

access private like member. Data member is accessed by specific pointer through qword to indicate its address.

Work citation:

http://www.ntu.edu.sg/home/ehchua/programming/cpp/cp4_pointerreference.html

https://www.geeksforgeeks.org/structure-member-alignment-padding-and-data-packing/