Brian Aguirre
ba5bx
inlab8.pdf

Parameter Passing:

Question 1:

In order to pass the different types either as values or references, the following general structured was used:

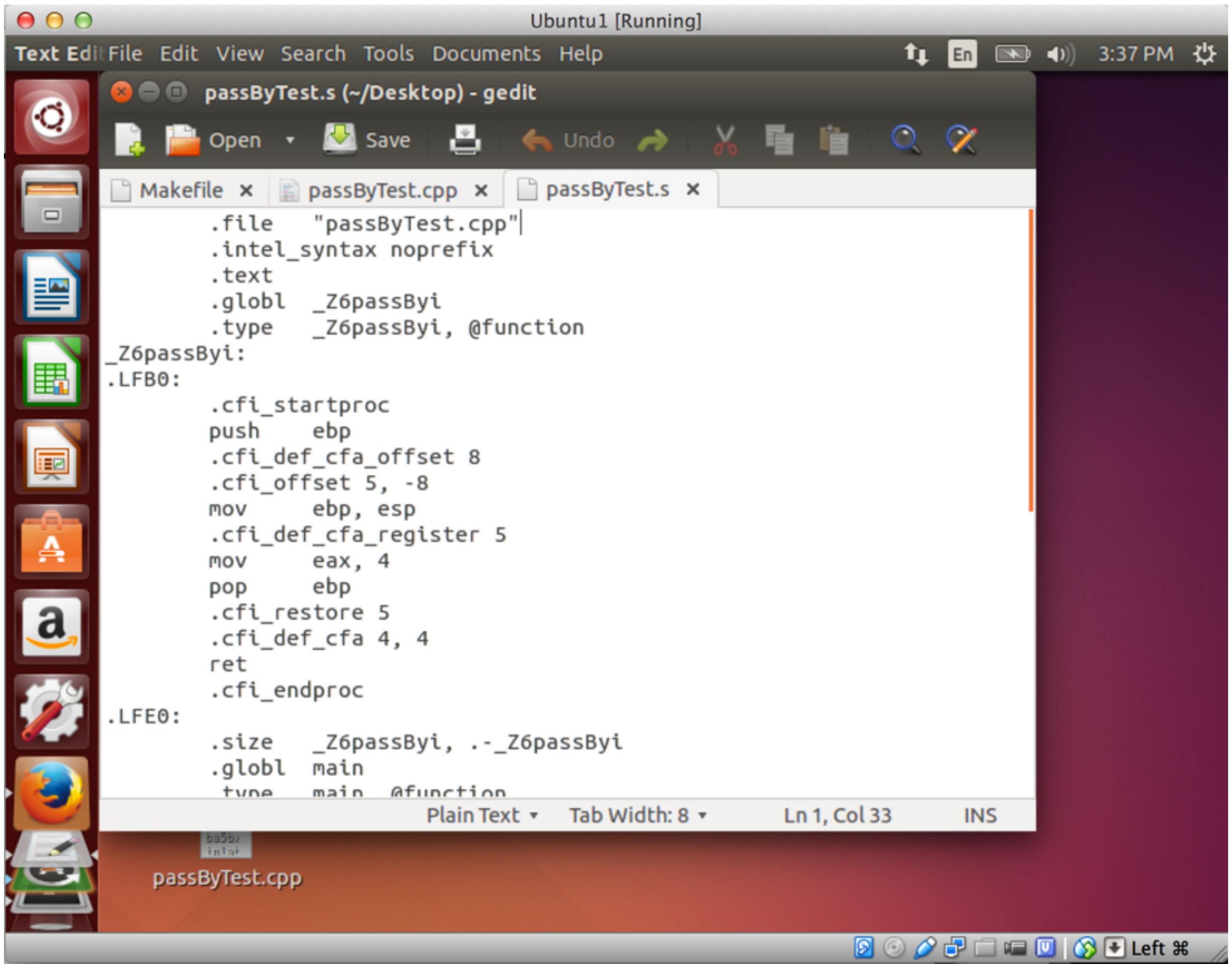
```
int passBy(type x){
        return 4;
}
```

and

```
int passBy(type &x){
        return 4;
}
```

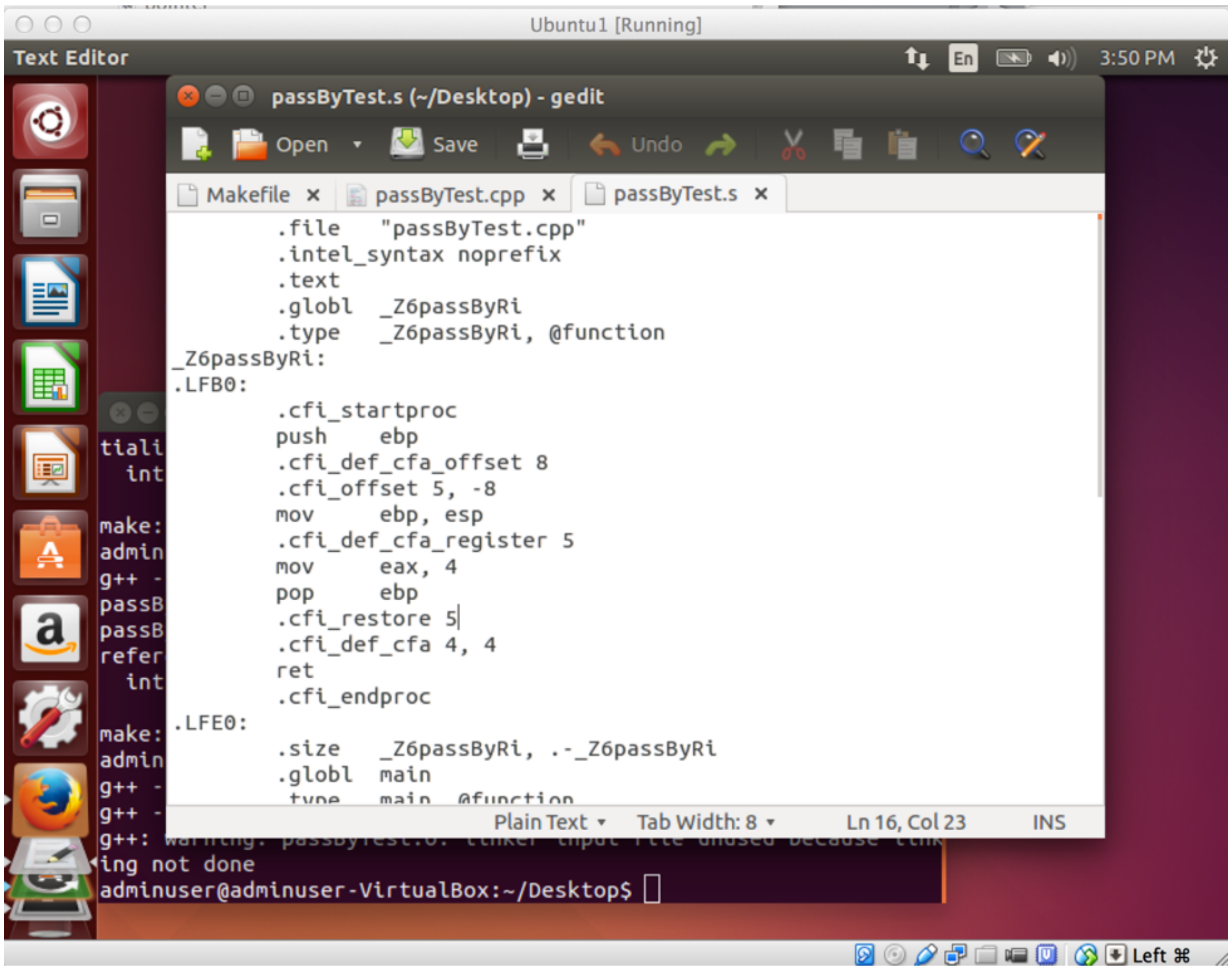
Type represents either an int, char, long, etc., while it is then viewed in assembly using the line from the lab: "-S -m32 -masm=intel" using g++.

Some of the screen captures generated from it are:
int passed by value:


Ubuntu1 [Running]
Text Edit File Edit View Search Tools Documents Help
3:37 PM
passByTest.s (~/Desktop) - gedit
Open Save Undo
Makefile passByTest.cpp passByTest.s
Plain Text Tab Width: 8 Ln 1, Col 33 INS
passByTest.cpp

```
        .file   "passByTest.cpp"
        .intel_syntax noprefix
        .text
        .globl  _Z6passByi
        .type   _Z6passByi, @function
_Z6passByi:
.LFB0:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        mov     eax, 4
        pop     ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   _Z6passByi, .-_Z6passByi
        .globl  main
        .type   main, @function
```

int passed by reference:



```
        .file   "passByTest.cpp"
        .intel_syntax noprefix
        .text
        .globl  _Z6passByRi
        .type   _Z6passByRi, @function
_Z6passByRi:
.LFB0:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        mov     eax, 4
        pop     ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   _Z6passByRi, .-_Z6passByRi
        .globl  main
        .type   main, @function
```

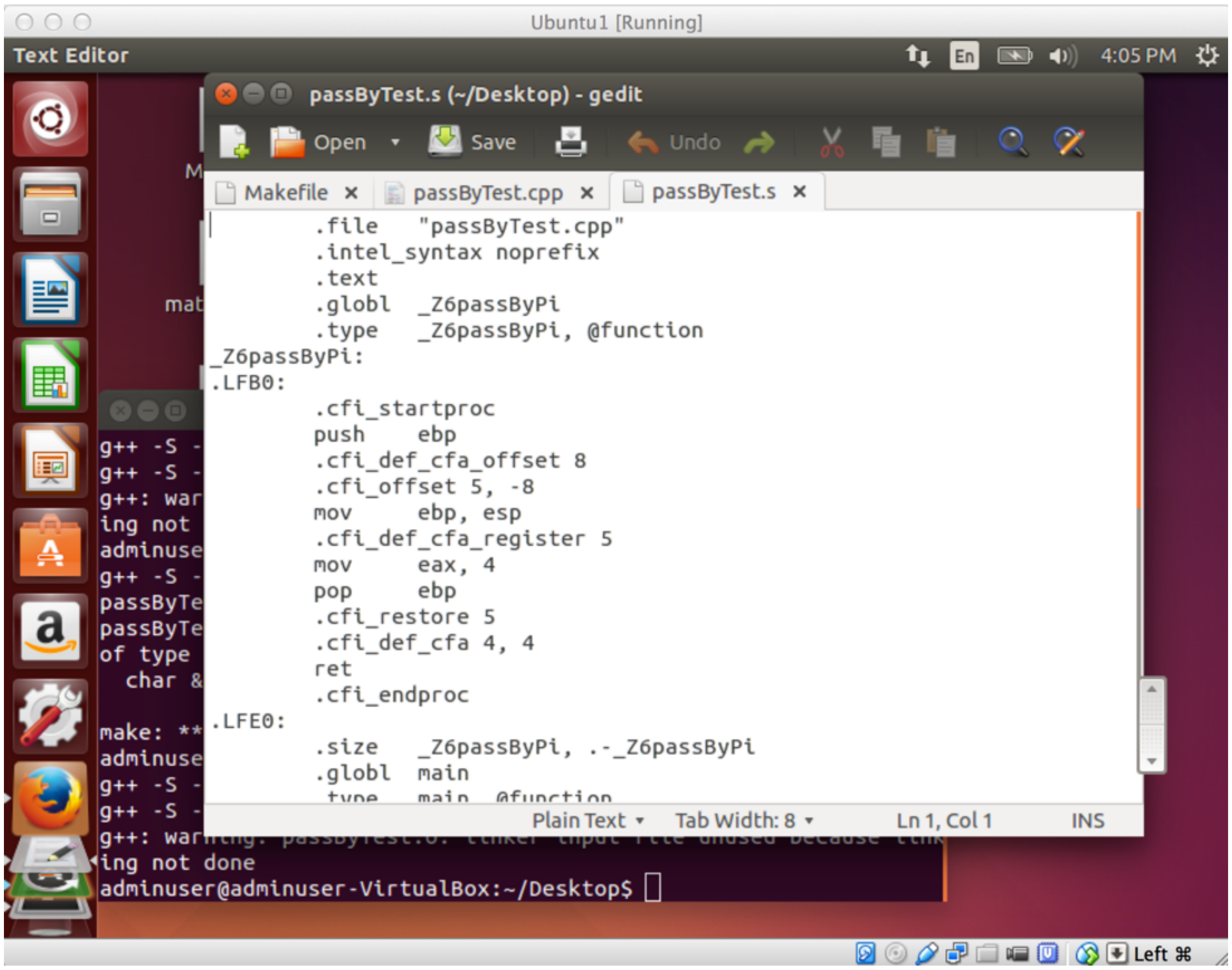
```
tiali
  int

make:
admin
g++ -
passB
passB
refer
  int

make:
admin
g++ -
g++ -
g++: warning: passByTest.o: linker input file unused because link
ing not done
adminuser@adminuser-VirtualBox:~/Desktop$
```

pointer passed by value:

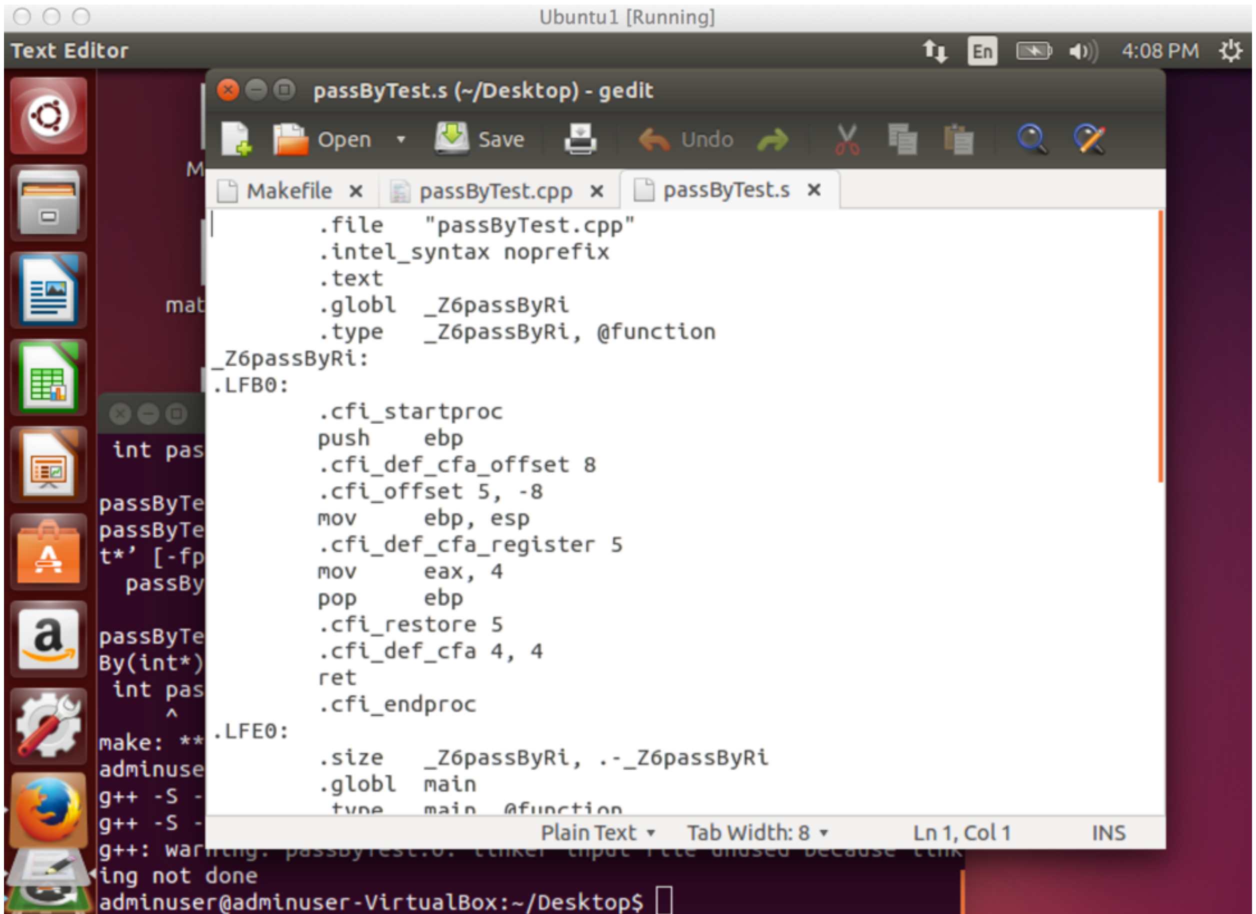
(Due to size, it is found on following page)



```
        .file   "passByTest.cpp"
        .intel_syntax noprefix
        .text
        .globl  _Z6passByPi
        .type   _Z6passByPi, @function
_Z6passByPi:
.LFB0:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        mov     eax, 4
        pop     ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   _Z6passByPi, .-_Z6passByPi
        .globl  main
        .type   main, @function
```

```
ing not done
adminuser@adminuser-VirtualBox:~/Desktop$
```

pointer passed by reference:



```
        .file   "passByTest.cpp"
        .intel_syntax noprefix
        .text
        .globl  _Z6passByRi
        .type   _Z6passByRi, @function
_Z6passByRi:
.LFB0:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        mov     eax, 4
        pop     ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   _Z6passByRi, .-_Z6passByRi
        .globl  main
        .type   main, @function
```

```
ing not done
adminuser@adminuser-VirtualBox:~/Desktop$
```

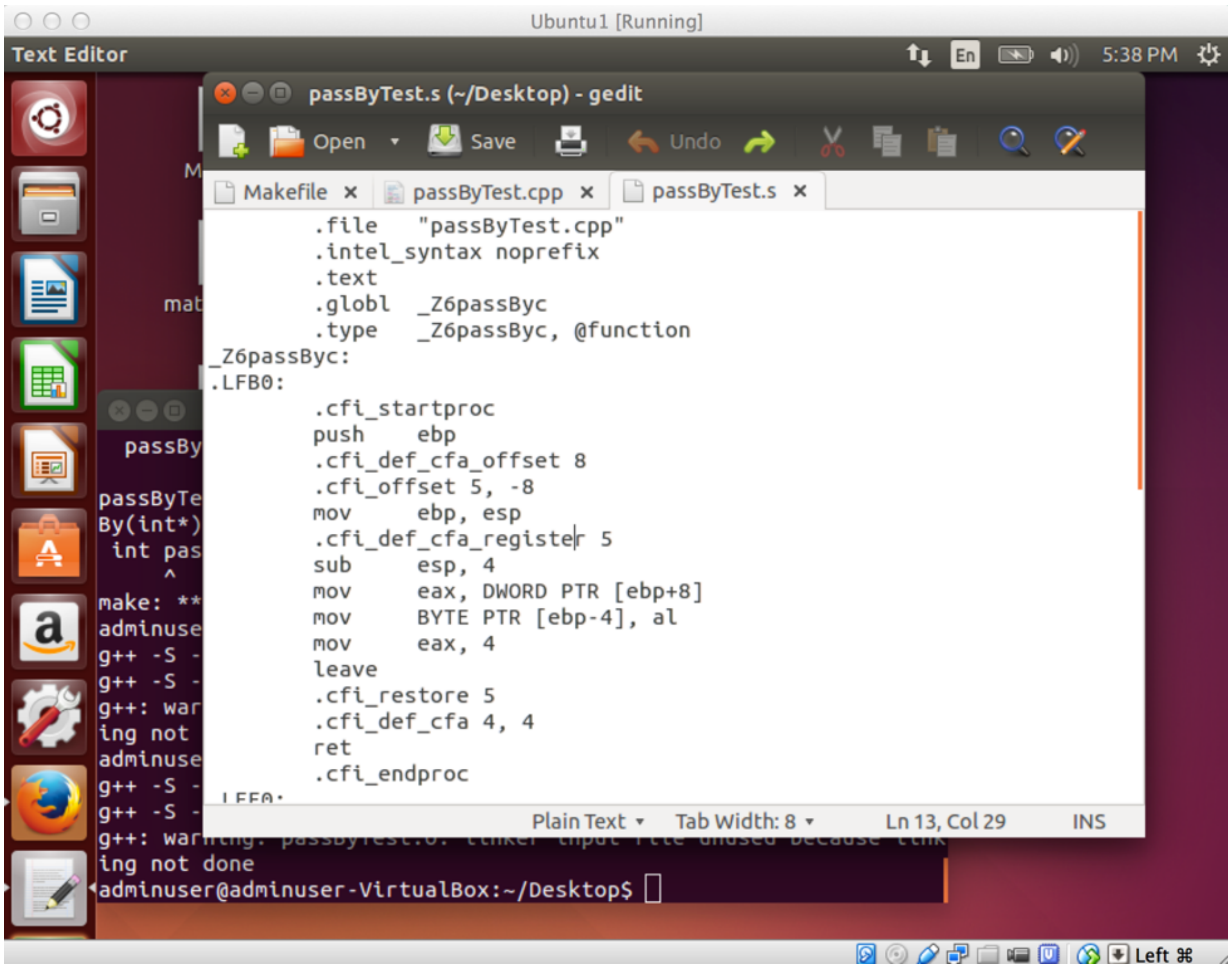
Based on these two examples, there is a overarching similarity within all of the parameters, whether they're passed by value or by reference. The assembly that appears repeatedly is as follows:

```
push    ebp
…
mov     ebp, esp
…
mov     eax, 4
pop     ebp
…
ret
```

The above was true for all of them except for char, which goes to show that despite the types, due to the size needed to represent these different types and lack of change of address for the values within the subroutine (meaning the pointer where the value was stored), there is no need for the values to differ and so the assembly for all of the types, when passed by value or reference is the same.

As mentioned above, char was different when passed by value. The following screen shot demonstrates what the assembly output was for the code:

```
int passBy(char x){
        return 4;
}
```



```
        .file   "passByTest.cpp"
        .intel_syntax noprefix
        .text
        .globl  _Z6passByc
        .type   _Z6passByc, @function
_Z6passByc:
.LFB0:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 4
        mov     eax, DWORD PTR [ebp+8]
        mov     BYTE PTR [ebp-4], al
        mov     eax, 4
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

```
ing not done
adminuser@adminuser-VirtualBox:~/Desktop$ 
```

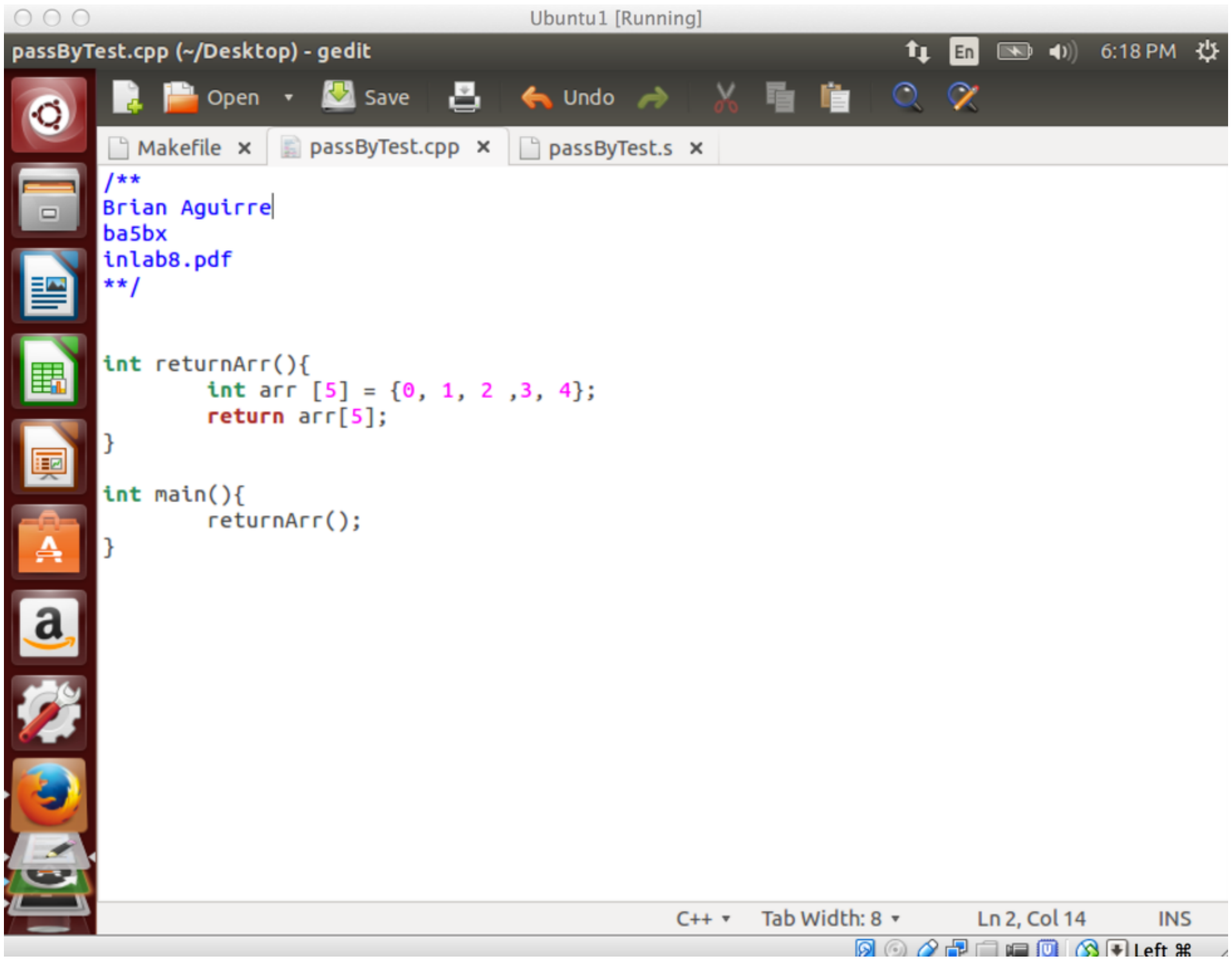
The following commands can be gathered from the previous screen shot:

```
push    ebp
…
mov     ebp, esp
…
sub     esp, 4
mov     eax, DWORD PTR [ebp+8]
mov     BYTE PTR [ebp-4], al
mov     eax, 4
…
ret
```

Here, the number 4 is processed Since char types are 1 byte, the stack pointer is decreased by 4 is first read as DWORD type then after that, it is decreased in size since it doesn't need to use as much memory for the char type, then last, it is moved back.

Question 2:

The screen capture below shows the code written in order to generate the assembly code of a function that contains an array, from which a value is returned:

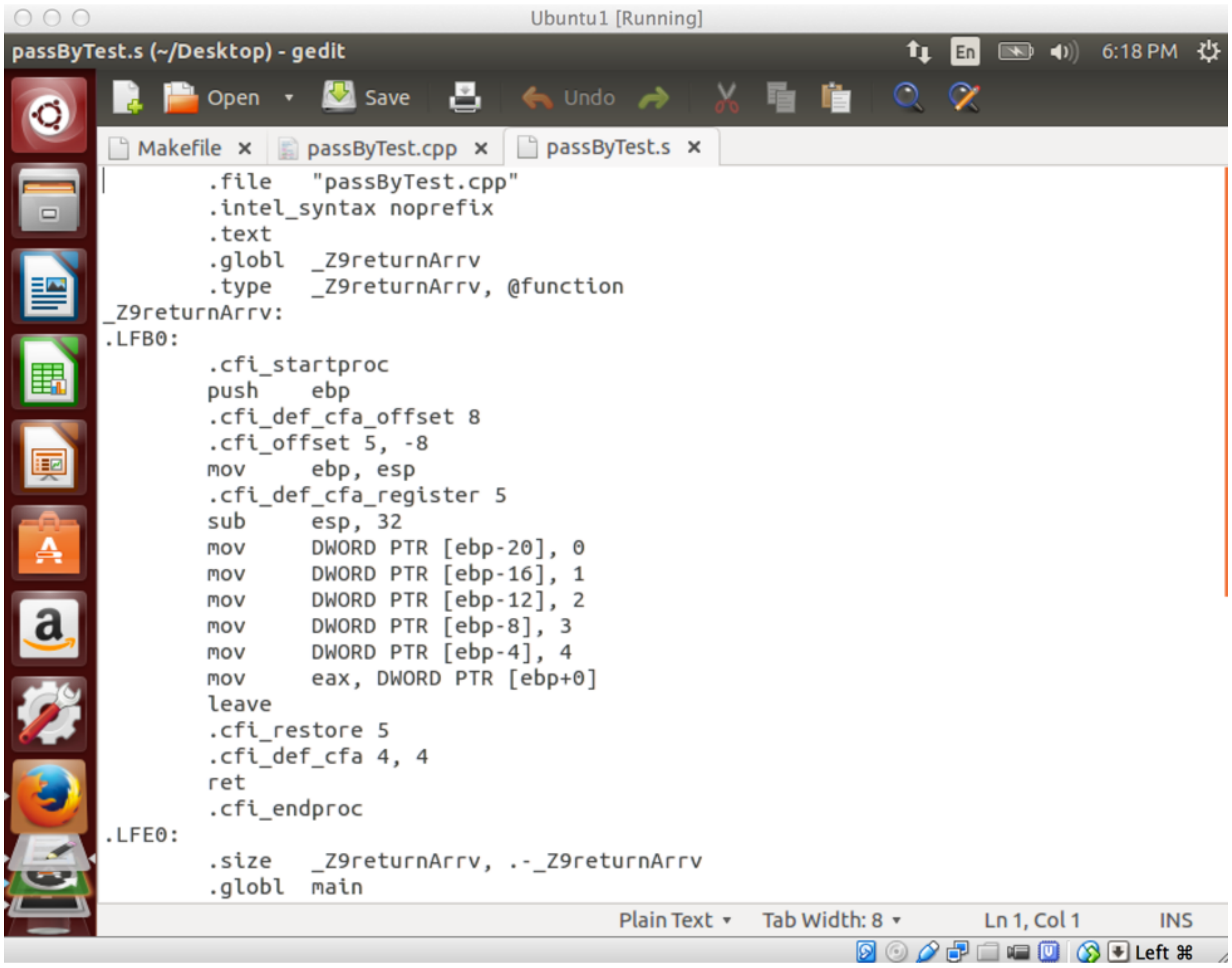


```
/**
Brian Aguirre
ba5bx
inlab8.pdf
**/


int returnArr(){
        int arr [5] = {0, 1, 2 ,3, 4};
        return arr[5];
}

int main(){
        returnArr();
}
```

The screen capture below shows the result of the generated assembly lines from the code above:



```
        .file   "passByTest.cpp"
        .intel_syntax noprefix
        .text
        .globl  _Z9returnArrv
        .type   _Z9returnArrv, @function
_Z9returnArrv:
.LFB0:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 32
        mov     DWORD PTR [ebp-20], 0
        mov     DWORD PTR [ebp-16], 1
        mov     DWORD PTR [ebp-12], 2
        mov     DWORD PTR [ebp-8], 3
        mov     DWORD PTR [ebp-4], 4
        mov     eax, DWORD PTR [ebp+0]
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   _Z9returnArrv, .-_Z9returnArrv
        .globl  main
```

The cpp files contains a method in which an array of 5 members is defined then the 5th element is returned. When this is processed through into assembly code there are a few important steps that should be pointed out. First, the pointer to where the array starts is located on the line “sub esp 32” where the stack pointer is decreased by 32. Then, as it can be seen by a couple of previous lines, there is a different of 8 bytes left as “offset” which explains why the first number, 0, is stored in ebp-20, and since each int type is given 4 bytes, the memory space decreased by four as the elements increase; 1 = ebp-16, 2 = ebp-12, etc.

Then the base address is stored in eax, and so the base of the Stack Pointer is moved there with the line “mov   eax, DWORD PTR [ebp+0]”.

Question 3

Through assembly, passing a type by pointer or reference, in this case it was an int, there was no difference. Assembly does both in similar manner, which means that the only difference between the two is really how the compiler understands pointer and references, which is by setting limits and requirements for references upon initializing and derefencing while those don’t really exist (to certain a extent) in pointers.