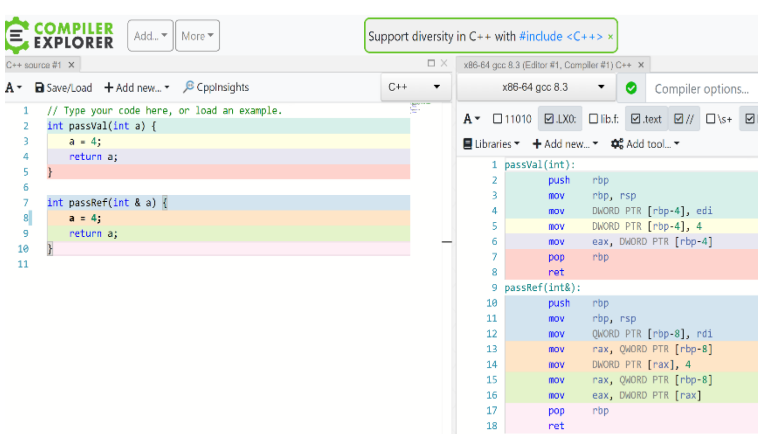
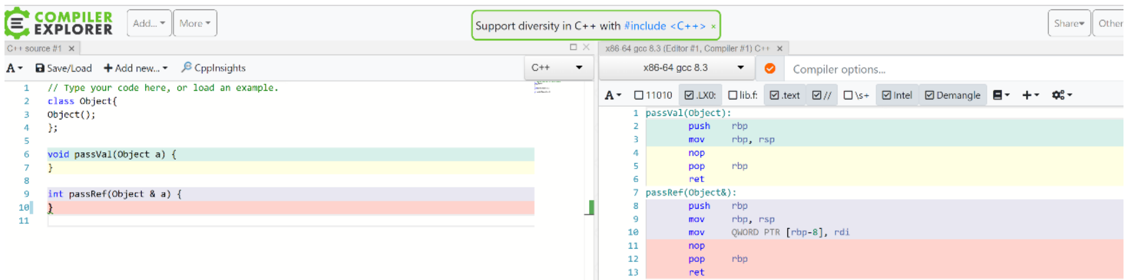
Eza Rasheed
er6qt
03-27-19
postlab8.pdf

Parameter Passing:

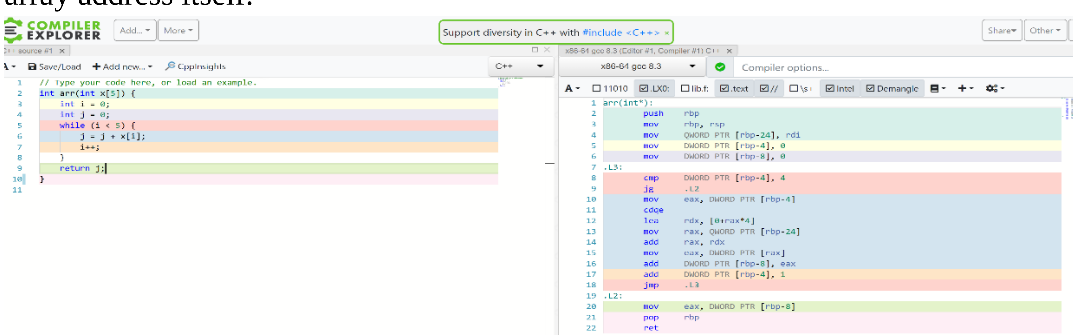
1. To demonstrate the difference in parameter registers when variables are passed by value vs. by reference, I created the simple program below in which the first takes in an integer parameter input passed by value and returns the value assigned to the int variable, and the second one passes it by reference. For some background information taken from an online article, when a parameter is passed by value, the caller and callee have two independent variables with the same value. If the callee modifies the parameter variable, the change is not noted to the caller. When a parameter is passed by reference, the caller and callee use the same variable for the parameter. If the callee modifies the parameter variable, the modification is noted to the caller's variable. Therefore, changes to variables are visible when you pass by reference. In C++, as mentioned in lecture, passing by value is when the original parameter is copied into a formal parameter and when pointers are passed in by value, and passing by reference(address) is when references are passed as parameters, which should allow the formal parameter to chance the value of the actual parameter argument. In the screenshot below, it can be seen that when I passed by value, edi is stored in memory address [rbp-4] and for pass by reference, rdi is stored in [rbp-8]. Passing by value took the returned number 4 and stored it at the memory address [rbp-4] whereas passing by reference uses the return register rax to store the value 4. The returned value in rax becomes QWORD PTR [rbp-8], which lets it store the memory address. Also to be noted is that in pass by value, the variable is stored as a DWORD and initializes the variable "a" to equal 4. In pass by reference, the variable is stored as a QWORD. This is because the registers are storing memory addresses, not just the value of the variable. In conclusion, it can be seen that passing by value makes a copy of the original parameters, whereas passing be reference stores a copy of the memory address of the original parameter.



2. In my simple function below, I take in an object passed by value and passed by reference. When you think about it, passing in an object is the same as passing in an int, except the stored memory address depends on what is in the object. Objects are passed by value in C++ when a function needs a copy of its arguments/parameters in memory and you want to do things to the object within the function without affecting the original. As can be seen in the register, when “a” is passed by value, rsp is moved into rbp (mov rbp, rsp). Contrastingly, though, when “a” is passed by reference, the value in “a’s” memory address is stored in memory (mov QWORD PTR [rbp-8], rdi).

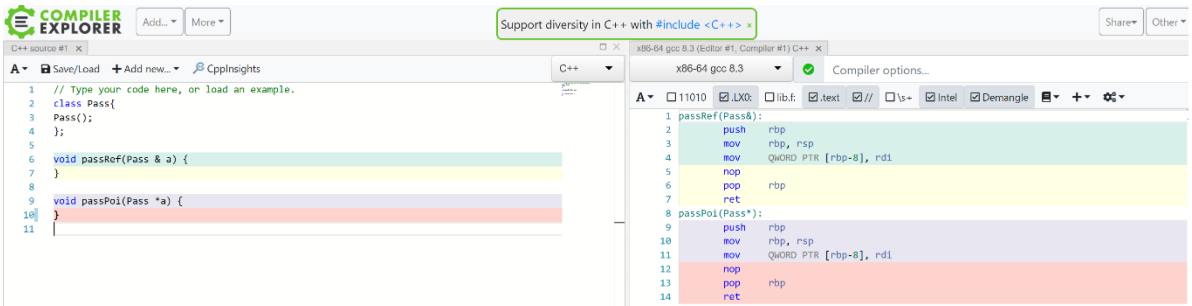


3. In my picture below, I created a function which takes in an array as a parameter. The while loop is how the array is passed into the function and how the callee accesses the parameters as it iterates through each value. An array is stored into memory as values and is passed by reference as the address of the array is stored in memory. These values are then accessed by a pointer which starts at the beginning of the array (at the first element) and then increments using the loop([0+rax*4]) to get the memory location of the target position(value). Since the memory address of each value in an array is a pointer, the dereferencing of that memory has to occur to access the array itself. The values in the array themselves are passed by value and stored in memory. A register-relative address is in which the data can be stored is line 20, where DWORD PTR [rbp-8] is being stored/moved into eax. The memory address of the array is being stored in memory based on the address in the register and proximity to that address in memory. We can then dereference the address by calling the register address that has the changes to access they array address itself.



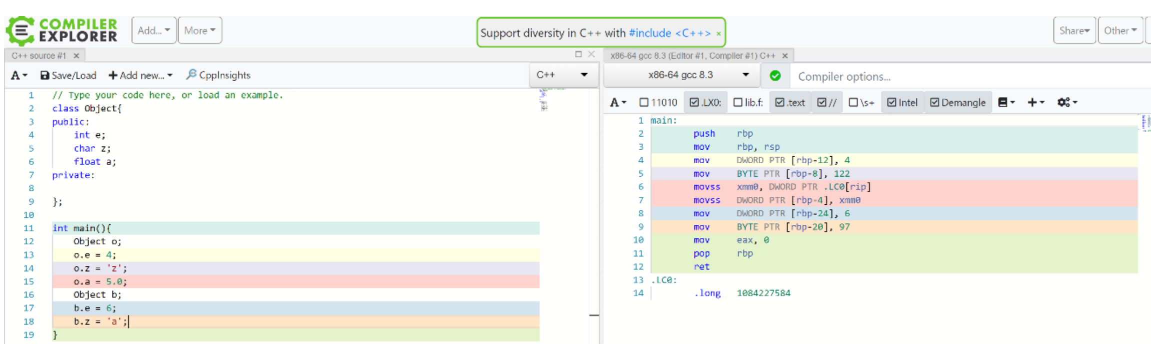
4. Passing values by reference is not different than passing by pointer, as can be seen in my assembly code below. When you pass by reference, the parameter storing value “a” will be allocated the same way as a pointer. “Passing by reference performs parameter passing using memory addresses of caller-saved registers, rather than values inside it.” Also, it can be noted that when passing by reference, assembly uses pointers (* for

dereferencing). Therefore, passing by reference and passing by pointer are quite identical. Looking at my screenshot below, you can observe that pointers store memory addresses the same way as passing by reference does; the memory address of “a” is stored in memory when passing by address (reference &) and pointer(*).



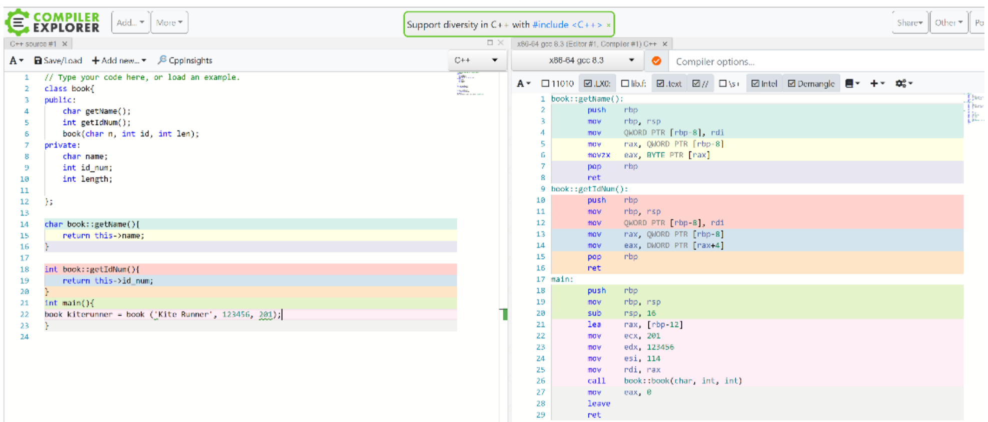
Objects:

1. I made a class called object below which tests how values are stored in different data types. Object data is laid out in memory according to the memory address. The fields are stored in a sequential order, one by one, depending on their size of bytes. Where it is stored is different based on the fact that calculations for the memory address locations are different. In my code, an integer is stored and then a char is stored, which leaves extra space after the char, until another integer is stored. This is because an integer is 4 bytes and a char is 1 byte, so after the char is stored, it is not at the right memory address. To accommodate for this, padding is produced so that the integer is stored at a valid address as can be seen in the code below in line 4-8. This makes sense why the memory addresses are different; different fields of different types have different sizes, so they are implemented based on the size in memory they take up. From looking at my code, I believe that C++ keeps different fields of an object together because they are stored in a stack. They are only a few bits away in distance on the stack, so they are usually together. To further my hypothesis, I did some research online. It mentions that C++ can keep different field together in one class “based on accessibility within the scope of the object. Depending on what was loaded into the register when initializing, the assembler knows which data member to access (pointer for each data field).



2. In my code below, I created a book object with the parameters being the name of the book, the id number if it, and the length (number of pages) of it. First it gets the char

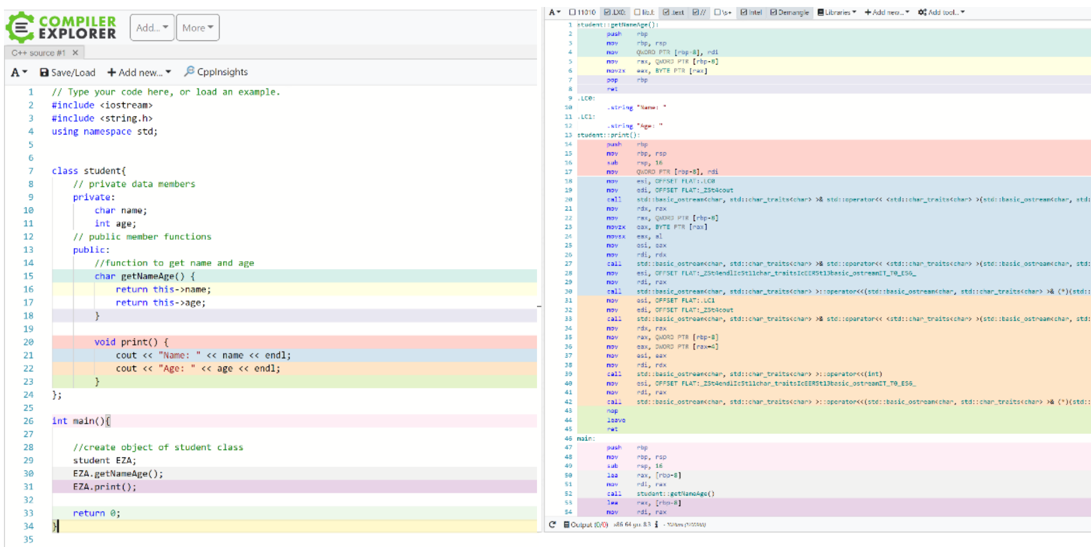
which is 1 byte and reserve 1 spot in memory to hold the char name. Then, it will read in the id and assign it a spot in memory after the char, meaning 4 spots as it is an int. Lastly, it will read in int length and assign it a spot after int id and int length's memory will be at that memory address. The public field in my code will allocate enough space(bytes) for the calls/rets. The assembly knows which data members to access based on the its memory address and size. I am not exactly sure how data fields are accessed, but from my research, it is stated that it is able to access the different fields for the objects through "the stack and it can use the order that fields were initialized to find them in the stack." Based off of this, I am assuming that assembly can find data fields from the register storing memory addresses and their values, which allows it to know the relation of different memory addresses compared to others. Examining the code below in its entirety, it can be seen that the differences in memory addresses are necessary in finding the data field's memory addresses and comparing them to one another to access a specific object field/value.



3. My hypothesis is that method invocation works for objects by looking at the local variables which all have a specific spot in the stack to know which object it is being called out of. The registers use the parameters (n, id, and len in the above code) to modify those specific parameters and return them as a result. In an article I read, they state that this is the whole purpose of the caller and callee in that the caller will call the callee to have the correct parameter values stored so that the method invocation can return the right value. Therefore, assembly knows which object it is being called out of based on the size of the object in memory and based off of its addresses. Adding on to the code above, I added another object to test the differences between the two different objects. As could be predicted, because I initialized the two objects the same way, the memory allocation was the same for both as they were the same size and contained the same parameters.

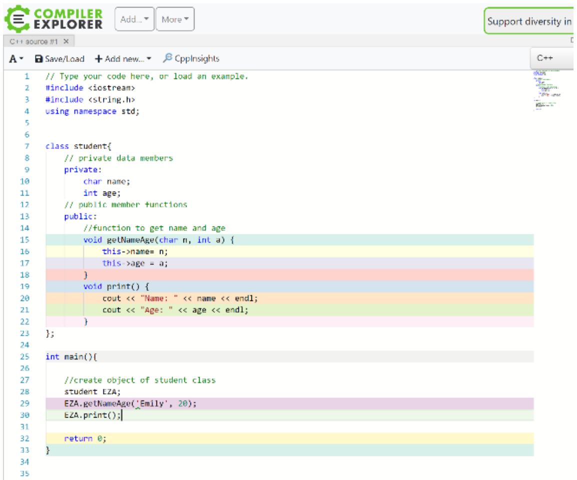
4. Private and public data members are accessed almost the same way in assembly by reading through what is inside the public and private fields and creating global functions that can be used in assembly. Although, the difference being, private member are accessible within the same class in which they are declared, while public members can access within the class and outside the class. For example, in my code below, I created a

main function that is able to access the public field functions (getName(), print()) outside of the function to set the functions; it can be seen that the public member functions are accessible within the main function by using the "." operator (i.e. EZA.getNameAge()). Data members are stored in order onto the stack in the order that they are pushed and popped and the registers are modified accordingly. Data members are accessed from inside and outside if they can take a field within a register and access/modify it to reach a result. Pointers can be created as I did in one of my public member functions to access that member using the registers, whether inside the function or outside, if they are already defined. Data members can be accessed from inside a member function by using them within a loop as the code runs when calling that specific function and modifying the parameters to use as arguments to those functions.



5. The public member functions accessed for my class are getNameAge(…) which returns a char and an int holding the person's name and age, respectively, and print(), which does not have any parameters as it just prints out the person's name and age. When the public member functions are accessed for the student class by getting called on an object (i.e. EZA.getNameAge('Emily', 20)), the "subroutine of the function is called by the caller" (main). Then, a "subroutine is made when the object is called on and it is stored before the main" so that it can be called. The memory address of the object is "stored into the first parameter passing register and then the callee function is called." The value returned from the callee function is moved into the return register eax so that it can be accessed in the caller function when the callee function is done; the registers are used so that they can store certain values to be returned in the end. The parameters are called by the standard parameter registers in order to call the function and edit the function, so in my case, it takes in parameters n and a in rbp and modifies it so that we get the correct return value. The "this" pointer points to the address of the object and the actual object is accessed using the square brackets where it is then moved and stored into a register (rax or edx). When you use the keyword "this", the "this" pointer is implemented so that "each object can get its own copy of the data member." It is accessed when you push it and move the

address of the object you are calling it on into the register, which stores the memory address for it. It is passed to member functions using the register itself just like other objects.



Works Cited

"Accessing Data Members of Class in C++." *Studytonight*, www.studytonight.com/cpp/accessing-data-members.php.

Data Members and Member Functions in C++ Programming, www.includehelp.com/cpp-tutorial/data-members-and-member-functions.aspx.

Dragonion, et al. "Visual C++ Inline x86 Assembly: Accessing 'This' Pointer." *Stack Overflow*, stackoverflow.com/questions/12194083/visual-c-inline-x86-assembly-accessing-this-pointer.

How to Pass and Return Object from a Function in C++?, www.programiz.com/cpp-programming/pass-return-object-function.

"Popular @." *How to Find in a Method Being Called, Which Class & Method Invoked It?*, www.javalobby.org/java/forums/t67019.html.

prakashprakash 30.4k2485112, et al. "What Are the Differences between a Pointer Variable and a Reference Variable in C++?" *Stack Overflow*, stackoverflow.com/questions/57483/what-are-the-differences-between-a-pointer-variable-and-a-reference-variable-in.

RoddyRoddy 46.1k35146241, et al. "Why Would You Pass an Object by Value in C++." *Stack Overflow*, stackoverflow.com/questions/11905275/why-would-you-pass-an-object-by-value-in-c.

"'This' Pointer in C++." *GeeksforGeeks*, 28 Sept. 2018, www.geeksforgeeks.org/this-pointer-in-c/.