X86 STRUCTS

James Riley Dorough

Liberty University

April 16th, 2021

Understanding structs is a useful tool when programing in any language. Additionally, understanding the difference between structs and classes is also keenly important. This paper covers the functionality and composition of structs in the x86 assembly language. This is demonstrated in the inclusion of images outlining a short x86 program. The code is broken into the initial setup section where the external libraries are included and the struct is defined. The second section contains the operations and functions that access the values in the struct. Lastly, the output of the code will tie together all the preparation and visualize the operation of the functions.

Firstly, we must discuss how the structs used in this lab differ from classes which object-oriented programmers may be more familiar with. Boccara (2017) conveys that the levels of abstraction provided by a struct are not as high as a class. Classes are intended to hide and operate on data members. Meanwhile, structs simply “bundle” data members together to provide simpler access for operations (Boccara, 2017). This allows for easier access to locations in memory. However, external functions or operations on the data must be included in the later program.

Graphical user interface, text

Description automatically generated

The program begins with an inclusion of external libraries to utilize the C printf functionality. Next, I set up the format specifiers that I need to print the various struct elements. I created one for printing a decimal integer with a line break, a string with a line break, one to print consecutive integers with a line feed to prepare for the next input, and lastly a new line alone. The next sections are where I reserved the memory for each struct element; requirement ID, requirement priority, team assigned, sub requirement array, and due date. After this I assigned values to each of these elements. Once I was finished with these elements, I closed them with the end commands.

Text

Description automatically generated

At this point in the code the struct has been defined and initialized. Now it is time to utilize it. The main label designates the start of a session of print commands. The program checks the struct memory location and prints the value loaded into the *rsi* register. Depending on the data type, the format specifier is loaded into the *rdi* register. To access data not located at the start of the struct, an offset is added to the memory to be printed. To print the array, the *rbx* register is used to store an integer and iterate through the values up to three. The *ret* return command is used to continue through the main section after the *iterate* function is called and completed. Finally, the code prepares to exit the code program gracefully.

Text

Description automatically generated

The program is assembled and set to an executable format to be run by bash. As seen above, the output matches the values specified earlier and exists without error. We see that each element is printed in the correctly specified format and does so on its own element line.

**References**

Boccara, J. (2017, June 13). *The real difference between struct and class*. Fluent C++. https://www.fluentcpp.com/2017/06/13/the-real-difference-between-struct-class/