CSC 3210-003 – Stack Machine Lab (Week 3)
Due Monday February 5, 2024 at 11:59 PM

Introduction

The purpose of this assignment is to design, code, and test a machine within the confines of the C programming language. The machine will consist of a data structure for storing data and operations that can be done against the data structure to perform calculations. While not strictly an "operating systems" exercise, it is intended to familiarize you with the development process on Linux and provide additional practice with dynamic memory.

Work on the assignment is to be done ***individually***. You are welcome to collaborate with class members, but the project must be your own work.

Prerequisites

Your Linux environment should be operational, and you should have a development environment (programmer's editor or IDE) setup and tested (i.e. you have written a hello world program). This assignment will also make use of the program 'valgrind' which may need to be installed. You can do this at the command line with the commands:

sudo apt update

sudo apt install valgrind

Background and References

A stack machine is a type of computer architecture in which values are stored on a stack rather than in general purpose registers. Values can be pushed from memory onto the stack and/or popped from the stack and stored into memory. Numerical operations are performed on items currently on the stack. For example, to execute an 'add' operation, the two operands are first pushed onto the stack. Then the 'add' instruction is executed which pops the values off the stack, adds them, and the result is pushed back on top of the stack.

While it might seem that using general purpose registers makes for a more flexible processor architecture, there are advantages of the simplicity of a stack machine. For example, there is no need for hardware to decode and access the register file, all operations are performed against the top of the stack. In fact, the Java Virtual Machine (JVM) is implemented in part as a stack machine.

Project Description

For this assignment you will be creating a program that can perform operations using a stack machine. The machine has no memory so all values for computation are stored entirely on the stack. Typically, a stack machine interprets instructions through machine code that has been compiled for the architecture, however, our stack machine will use function calls to perform operations.

The stack is a first-in-last-out data structure. Items are pushed (inserted) onto the top of the stack and popped (removed) from the top of the stack. The first thing pushed into the stack is therefore the last thing that can be removed. Think of it like stacking blocks on top of each other. You can't move the bottom block without first removing all the blocks on top of it.

Beyond push and pop, our stack machine can perform mathematical operations add, subtract, multiply, divide, as well as a special operation called rotate which rotates elements on the stack. The size (number of elements it can hold) of the stack is unbounded, so you must be able to dynamically allocate and free elements as needed.

CSC 3210-003 – Stack Machine Lab (Week 3)
Due Monday February 5, 2024 at 11:59 PM

Development Requirements

A header file is supplied that represents the "public" interface that you must implement. Each function is documented within the head file. You are ***NOT*** allowed to change this header file in any way. You should supply ***ONE*** source file with the implementation for all functions (stackm.c).

Upon reviewing the header file, you may notice some concerns:

- First, the functions in this "public" interface only manage the stack's memory and organization. The user has full access to the internals. Of course, some of this is an artifact of the C language that does not include the protections provided by languages such as Java and C++.
- Second, note that the user must pass a pointer to every stack function. Again, a necessity due to not using an object-oriented language.

In addition to the stack machine implementation, supply a "driver" that will completely test your stack machine implementation. Here is an example test driver, however it is not all-inclusive.

```
#include <stdio.h>
#include "stackm.h"

int main(void) {

   // Initialize the stack
   struct stackm_t my_stack;
   sm_init(&my_stack);

   // Test pushing of values
   sm_push(&my_stack, 2);
   sm_push(&my_stack, 3);
   sm_push(&my_stack, 4);
   sm_print(&my_stack);

   // Test popping of values
   sm_pop(&my_stack);
   sm_print(&my_stack);

   // Test retrieving the top
   int value = 0;
   sm_top(&my_stack, &value);
   printf("%d\n", value);

   // Test clear
   sm_clear(&my_stack);
```

```
    sm_print(&my_stack);

    // Test addition
    sm_push(&my_stack, 2);
    sm_push(&my_stack, 3);
    sm_print(&my_stack);
    sm_add(&my_stack);
    sm_print(&my_stack);
    sm_pop(&my_stack);

    // Test subtraction
    sm_push(&my_stack, 10);
    sm_push(&my_stack, 5);
    sm_print(&my_stack);
    sm_sub(&my_stack);
    sm_print(&my_stack);
    sm_pop(&my_stack);

    // Test multiplication
    sm_push(&my_stack, 10);
    sm_push(&my_stack, 11);
    sm_print(&my_stack);
    sm_mult(&my_stack);
    sm_print(&my_stack);
    sm_pop(&my_stack);

    // Test division
    sm_push(&my_stack, 10);
    sm_push(&my_stack, 2);
    sm_print(&my_stack);
    sm_div(&my_stack);
    sm_print(&my_stack);
    sm_pop(&my_stack);

    // Test rotate
    sm_push(&my_stack, 10);
    sm_push(&my_stack, 11);
    sm_push(&my_stack, 12);
    sm_push(&my_stack, 13);
    sm_push(&my_stack, 14);
    sm_push(&my_stack, 15);
```

```
    sm_print(&my_stack);
    sm_rotate(&my_stack, 5);
    sm_print(&my_stack);

    return 0;
}
```

The idea is that a user of the stack machine will create a stack structure, then call sm_init to initialize the stack. After the stack is initialized, additional operations can be performed using the functions.

Additional Development Requirements

- Do ***NOT*** include source files inside each other via #include. The #include preprocessor macro should only be used for header files
- You ***must*** put all source code in a src directory.
- You ***must*** submit a make file in your src directory that will build ***ALL*** executables by just typing make on the command line
    - Research make files on how to correctly create more than one executable from a single make file.

Code Structure

Code must follow style guidelines as defined in the course material

Testing and Debugging

Test Cases

As stated above, you must write a driver that completely tests your stack machine. So what is complete testing? How do you go about this? You will have to put some thought into this and use your experience to guide you. It would make sense that you will have to call every method at least once, probably many times under different conditions. Look for boundary cases. For example, be sure to test sm_pop() with an empty stack as well as a non-empty stack.
NOTE: You can assume the user will ***ALWAYS*** invoke sm_init before using the stack for any other function.

Debugging

See the course debugging tips for using gdb and valgrind to help with debugging.
Your program must be free of run-time errors and memory leaks.

Deliverables

When you are ready to submit your assignment prepare your repository:

- Make sure your name, assignment name, and section number are all files in your submission – in comment block of source file(s) and/or at the top of your report file(s)
- Make sure all source and header files are in your src directory only
- Make sure you have completed all activities and answered all questions
- Make sure you cite your sources for all research
- Make sure your assignment code is commented thoroughly and follows the coding standards

- Include in your submission, a set of suggestions for improvement and/or what you enjoyed about this assignment
- Make sure all files are zipped and uploaded into Canvas

Additional Submission Notes

If/when using resources from material outside what was presented in class (e.g., Google search, Stack Overflow, etc.) document the resource used in your submission. Include exact URLs for web pages where appropriate.

NOTE: Sources that are not original research and/or unreliable sources are not to be used. For example:

- Wikipedia is not a reliable source, nor does it present original research: https://en.wikipedia.org/wiki/Wikipedia:Wikipedia_is_not_a_reliable_source
- ChatGPT is not a reliable source: https://thecodebytes.com/is-chatgpt-reliable-heres-why-its-not/

You should ensure that your program compiles without warnings (using -Wall) or errors prior to submitting. Make sure your code is well documented (inline comments for complex code as well as comment blocks on functions). Make sure you check your code against the style guidelines.

Grading Criteria

- (5 Points) Submitted files follow submission guidelines
  - Only the requested files were submitted
  - Files are contain name, assignment, section
  - Sources outside of course material are cited
- (5 Points) Suggestions
  - List of suggestions for improvement and/or what you enjoyed about this assignment
- (10 Points) Code standard
  - Code compiles without errors or warnings
  - Code is formatted and commented per the course coding standard
- (5 Points) Memory management
  - Code contains no memory leaks
- (5 Points) Build - Code compiles without warnings or errors
- (20 Points) Test Cases - Thoroughness of submitted test driver
- (50 Points) Instructor Tests - Implementation passes all instructor test cases