Computing Machinery I
Assignment 5

Part A: Global Variables and Separate Compilation

A stack data structure can be implemented using an array, as shown in the following C program:

```
#include <stdio.h>
#include <stdlib.h>

#define STACKSIZE   5
#define FALSE       0
#define TRUE        1

/* Function Prototypes */
void push(int value);
int pop();
int stackFull();
int stackEmpty();
void display();

/* Global Variables */
int stack[STACKSIZE];
int top = -1;

int main()
{
    int operation, value;

    do {
        system("clear");
        printf("### Stack Operations ###\n\n");
        printf("Press 1 - Push, 2 - Pop, 3 - Display, 4 - Exit\n");
        printf("Your option? ");
        scanf("%d", &operation);
        switch (operation) {
            case 1:
                printf("\nEnter the positive integer value to be pushed: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                value = pop();
                if (value != -1)
                    printf("\nPopped value is %d\n", value);
                break;
            case 3:
                display();
                break;
            case 4:
                printf("\nTerminating program\n");
                exit(0);
            default:
                printf("\nInvalid option! Try again.\n");
                break;
        }
        printf("\nPress the return key to continue . . . ");
        getchar();
        getchar();
    } while (operation != 4);

    return 0;
}
```

```
void push(int value)
{
    if (stackFull())
        printf("\nStack overflow! Cannot push value onto stack.\n");
    else {
        stack[++top] = value;
    }
}

int pop()
{
    register int value;

    if (stackEmpty()) {
        printf("\nStack underflow! Cannot pop an empty stack.\n");
        return (-1);
    } else {
        value = stack[top];
        top--;
        return value;
    }
}

int stackFull()
{
    if (top == STACKSIZE - 1)
        return TRUE;
    else
        return FALSE;
}

int stackEmpty()
{
    if (top == -1)
        return TRUE;
    else
        return FALSE;
}

void display()
{
    register int i;

    if (stackEmpty())
        printf("\nEmpty stack\n");
    else {
        printf("\nCurrent stack contents:\n");
        for (i = top; i >= 0; i--) {
            printf("  %d", stack[i]);
            if (i == top) {
                printf(" <-- top of stack");
            }
            printf("\n");
        }
    }
}
```

Translate all functions except main() into ARMv8 assembly language, and put them into a separate assembly source code file called *a5a.asm*. These functions will be called from the main() function given above, which will be in its own C source code file called *a5aMain.c*. Also move the global variables into *a5a.asm*. Your assembly functions will call the printf() library routine. Be sure to handle the global variables and format strings in the appropriate way. Input will come from standard input. Run the program to show that it is working as expected, capturing its output using the *script* UNIX command, and name the output file *script1.txt*.

Part B: External Pointer Arrays and Command-Line Arguments

Given the following declaration in C:

```
char *month[] = {"January", "February", "March", "April", "May",
                 "June", "July", "August", "September", "October",
                 "November", "December"};
```

create an ARMv8 assembly language program to accept as command line arguments three strings representing a date in the format *mm dd yyyy*. Your program will print the date with the name of month as well as the correct suffix. For example:

```
./a5b 9 11 1990
September 11th, 1990
```

Be sure to use the proper suffix for the day of the month. For example, one should distinguish the 11th from the 1st, 21st, and 31st. Don't forget the comma after the day. Your program should exit, printing this error message, if the user does not supply three command-line arguments:

```
usage: a5b mm dd yyyy
```

You will need to call atoi() to convert strings to numbers, and printf() to produce the output. Be sure to do range checking for the day, month, and year. Name your source code file *a5b.asm*. Run your program three times with different input to illustrate that it works; capture the output using the *script* UNIX command. Name the output file *script2.txt*.

New Skills need for this Assignment:

- Understanding and use of external variables in SPARC assembly
- Separate compilation
- Calling assembly functions from main()
- Calling library functions from assembly routines
- External arrays of pointers
- Command line arguments

Submit the following:

1. Your source code and 2 scripts via electronic submission. Use the *Assignment 5* Dropbox Folder in D2L to submit electronically. Your TA will assemble and run your programs to test them. Name your files *a5aMain.c* and *a5a.asm* for Part A, and *a5b.asm* for Part B, and the scripts *as script1.txt* and *script2.txt*.

Computing Machinery I
Assignment 5 Grading

Student:_____

Part A:			
Correct use of external variable(s)	4	_____	
push() function in assembly	4	_____	
pop() function in assembly	4	_____	
stackFull() function in assembly	4	_____	
stackEmpty() function in assembly	4	_____	
display() function in assembly	8	_____	
Linking of separate source code modules	2	_____	
Correct manipulation of stack	4	_____	
Part B:			
Command line arguments	4	_____	
Correct use of external pointer arrays	4	_____	
Calls to library functions	2	_____	
Range checking	3	_____	
Correct output	4	_____	
2 Scripts showing I/O	4	_____	
Complete documentation and commenting	4	_____	
Design quality	4	_____	
Total	**63**	_____	_____%