

# Computing Machinery I

## Assignment 5

### Part A: Global Variables and Separate Compilation

Reverse Polish (or postfix) notation can be used to represent arithmetic expressions. For example, the infix expression

$(1 - 2) * (4 + 5) =$

is the following using reverse Polish notation:

$1\ 2\ -\ 4\ 5\ +\ *\ =$

Hewlett-Packard calculators often use this form of entry. The following is a C program that emulates such a calculator:

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

/* Constants */
#define MAXOP 20
#define NUMBER '0'
#define TOOBIG '9'

/* Function prototypes */
int push(int f);
int pop();
void clear();
int getop(char *s, int lim);
int getch();
void ungetch(int c);

int main()
{
    int type;
    char s[MAXOP];
    int op2;

    while ((type = getop(s, MAXOP)) != EOF) {
        switch (type) {
            case NUMBER:
                push(atoi(s));
                break;
            case '+':
                push(pop() + pop());
                break;
            case '*':
                push(pop() * pop());
                break;
            case '-':
                op2 = pop();
                push(pop() - op2);
                break;
            case '/':
                op2 = pop();
                if (op2 != 0)
                    push(pop() / op2);
                else
                    printf("zero divisor popped\n");
                break;
            case '=':
                printf("\n%d\n", push(pop()));
                break;
            case 'c':
                clear();
                break;
            case TOOBIG:
                printf("%.20s ... is too long\n", s);
                break;
        }
    }
}
```

```

        default:
            printf("unknown command %c\n", type);
            break;
    }
}

return 0;
}

#define MAXVAL 100

int sp = 0;
int val[MAXVAL];

int push(int f)
{
    if (sp < MAXVAL)
        return val[sp++] = f;
    else {
        printf("error: stack full\n");
        clear();
        return 0;
    }
}

int pop()
{
    if (sp > 0)
        return val[--sp];
    else {
        printf("error: stack empty\n");
        clear();
        return 0;
    }
}

void clear()
{
    sp = 0;
}

int gettop(char *s, int lim)
{
    int i, c;

    while ((c = getch()) == ' ' || c == '\t' || c == '\n')
        ;

    if (c < '0' || c > '9')
        return c;

    s[0] = c;
    for (i = 1; (c = getchar()) >= '0' && c <= '9'; i++)
        if (i < lim)
            s[i] = c;

    if (i < lim) {
        ungetch(c);
        s[i] = '\0';
        return NUMBER;
    } else {
        while (c != '\n' && c != EOF)
            c = getchar();
        s[lim-1] = '\0';
        return TOOBIG;
    }
}

#define BUFSIZE 100

char buf[BUFSIZE];
int bufp = 0;

```

```

int getch()
{
    return bufp > 0 ? buf[--bufp] : getchar();
}

void ungetch(int c)
{
    if (bufp > BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

```

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 library routines `printf()` and `getchar()`. Be sure to handle the global variables and format strings in the appropriate way. Input will come from standard input; the program is terminated by typing control-d. 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*. Use a variety of input expressions to show that your program is calculating correctly.

## 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 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*.

**Student:** \_\_\_\_\_

Correct use of external variable(s)	4	_____
push() function in assembly	4	_____
pop() function in assembly	4	_____
clear() function in assembly	2	_____
getop() function in assembly	6	_____
getch() function in assembly	3	_____
ungetch() function in assembly	3	_____
Linking of separate source code modules	2	_____
Correct evaluation of expressions	4	_____

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

Formatting (use of columns and white space)	4	
---	---	--

Design quality	4	
----------------	---	--

<b>Total</b>	<b>65</b>	<b>%</b>
--------------	-----------	----------