ECE391: Computer Systems Engineering — Spring 2021
Problem Set 1 — **Due: in Gitlab repository by 11:59:PM CST Monday 8 February (1:59PM in China on Tuesday 9 February)**

You may attempt this problem set in the class VM (devel) or on any EWS Linux computer.

Work in groups of at least four people. The person submitting the code **(and NO ONE ELSE IN THE GROUP)** should list all group members' netids, including their own, in the file `partners.txt`, with each netid on a separate line. For example, `partners.txt` should contain the following if submitted by `aamirh2`:

```
aamirh2
hagrawa3
ywu112
sjeon12
ch5
```

Note: if you want to attempt these problems on your own, leave `partners.txt` blank and the autograder will run on your repository but not update your score. The autograder will run two times before the deadline so you can track your progress.

WARNING: Any `partners.txt` files that conflict with any other `partners.txt` file will earn 0 pts for all concerned.

Problem 1: GDB Warm Up (5 pts)

Write the command(s) needed to achieve the following tasks in GDB.

1. Show the value of variable `test` in hex format.
2. Show the top four bytes on your stack word-by-word. The output should look something like "`0x0102 0x0304`," NOT "`0x01020304`."
3. Print all register values.
4. Set a breakpoint at line 391 in the file `ece.c`.
5. Connect to the test_(no)debug VM in the lab setup.

Write your solution in `p1/p1_soln.txt`, with answers to each question on a separate line.

Problem 2: Mapping C to Assembly (10 pts)

Write x86 assembly code for the body of the `kthSmallest` function found in `kthSmallest.c` (also provided below). Make sure to set up and tear down the stack frame as well as save and restore any callee-saved registers (if you use them). Include comments (but don't overdo it!) in your assembly code to show the correspondence between the C code and your x86 code.

- The `kthSmallest_asm` function in `kthsmallest_asm.S` is partially completed for you. You must complete the function.
- The `swap` function is given. Use it directly in your code (do not reimplement it).
- Make sure that your code and comments are easy to read. We reserve the right to take points off if your answer is too hard to follow.
- You must synthesize your answer without the help of a computer. For example, you may not use a compiler to assemble it. If you are caught doing so, you will receive a 0.
- You must **translate** the code. A functionally equivalent algorithm with a different structure will receive a 0.
- You must write your solution in `p2/kthsmallest_asm.S` and submit it through Git.

To build the code (no debug flag):

```
$ make clean && make
```

To run the code:

```
$ ./kthsmallest input_1.txt
```

To build the code (debug flag):

```
$ make clean && make debug
```

To run the code (debug):

```
$ gdb --args ./kthsmallest input_1.txt
```

```
**** Excerpt from p2/kthsmallest.c ****
// kthSmallest_c
//   Finds the kth smallest element in the given array.
int kthSmallest_c(int arr[], int left, int right, int k) {
    int pivot = arr[right];
    int i = left;
    for (int j=left; j<right; j++) {
        if (arr[j] <= pivot) {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[right]);

    if (k==i) {
        return arr[k];
    } else if (k < i) {
        return kthSmallest_c(arr, left, i-1, k);
    } else {
        return kthSmallest_c(arr, i+1, right, k);
    }
}
```

Problem 3: Mapping Assembly to C (10 pts)

Write a C function equivalent to the x86 assembly function, `mystery_asm` found in `mystery_asm.S` (also provided below).

- Make sure that your code and comments are easy to read. We reserve the right to take points off if your answer is too hard to follow.
- You must **translate** your code. A functionally equivalent algorithm with a different structure will receive a 0.
- Write your solution in `p3/mystery.c` and submit it through Git.

To build the code (no debug flag):

```
$ make clean && make
```

To run the code:

```
$ ./mystery ./input_1.txt
```

To build the code (debug flag):

```
$ make clean && make debug
```

To run the code (debug):

```
$ gdb --args ./mystery ./input_1.txt
```

```
**** Excerpt from mystery_asm.S ****
.text
.global mystery_asm

# Mystery (Assembly)
# Calculates the mystery function of the two input numbers
#
# Registers:
#   eax - For division op & Return Value
#   ebx - x
#   ecx - y
#   edx - For division op
#   edi - greater number

mystery_asm:
  pushl %ebp
  movl %esp, %ebp

  pushl %ebx
  pushl %edi

  movl 12(%ebp), %edi
  movl 12(%ebp), %ebx
  movl 8(%ebp), %ecx

  cmpl $0, %ebx
  jle invalid_input

  cmpl $0, %ecx
  jle invalid_input

  cmpl %ebx, %ecx
  je invalid_input
  jl op_loop

  movl 8(%ebp), %edi

op_loop:
  xorl %edx, %edx
  movl %edi, %eax
  idiv %ebx
  cmpl $0, %edx
  jne repeat

  xorl %edx, %edx
  movl %edi, %eax
  idiv %ecx
  cmpl $0, %edx
  jne repeat

  movl %edi, %eax
  jmp finish

repeat:
  incl %edi
  jmp op_loop

invalid_input:
  movl $-1, %eax

finish:
  popl %edi
  popl %ebx
  leave
  ret
```