CSci 2021, Fall 2018 Written Exercise Set 3

Due: Monday, November 12th at beginning of lecture

Submit this assignment on paper at **the beginning** of your lecture section. We strongly recommend that you type and print out your solutions. Please label your assignment with your name, UMN email address, and the time of your recitation section (10:10, 11:15, 12:20, or 1:25).

Problem 1: Arrays in Assembly

In this problem, the C code for a main function that calls the function `matrix_calculation` is provided below. The function `matrix_calculation` has its assembly given below that. Based on the output we have given you from `main`, it is your job to figure out the values you can fill in for `a`, `b`, and `c` in `my_span` that lead to that output being produced. You can write your answer as "a = value, b = value, ...". In the process of figuring this out, you may find it helpful to write C code for `matrix_calculation`. Writing C code is not required, and you can get full credit for just the correct values, but if you turn in C code for `matrix_calculation`, it will increase the possibility of getting partial credit.

```
int main(int argc, char **argv) {
  int my_span[3][3] = {{1,1,1},{a,b,-5},{2,11,c}};

  int result_val = matrix_calculation(my_span, 1, 2, 3);

  printf("result_val is %d\n", result_val);
}
```

```
matrix_calculation:
        pushq   %rbx
        movl    $0, %r10d
        movl    $0, %eax
        jmp     .L2
.L3:
        movslq  %esi, %r8
        leaq    (%r8,%r8,2), %r9
        leaq    0(,%r9,4), %r8
        addq    %rdi, %r8
        movslq  %r10d, %r11
        movslq  %edx, %r9
        leaq    (%r9,%r9,2), %rbx
        leaq    0(,%rbx,4), %r9
        addq    %rdi, %r9
        movl    (%r9,%r11,4), %r9d
        imull   (%r8,%r11,4), %r9d
        addl    %r9d, %eax
        addl    $1, %r10d
.L2:
        cmpl    %ecx, %r10d
        jl      .L3
        popq    %rbx
        ret
```

The given output for main() is: `result_val is 42`

Problem 2: Translate to Y86-64

The given C code is for a function that checks if all the elements of a linked list add up to 100.

```
struct linked_list{
    long val;
    struct linked_list *next;
};

int sum_to_100(struct linked_list *start){
    struct linked_list *current_node = start;
    long total = 0;
    while( current_node != NULL ){
        total += current_node->val;
        current_node = current_node->next;
    }
    if( total == 100 ){
        return 1;
    }
    else{
        return 0;
    }
}
```

The x86-64 assembly given below was compiled with GCC from the C code given above.

```
sum_to_100:
        movq        $0, %rax
        jmp         .L2
.L3:
        addq        (%rdi), %rax
        movq        8(%rdi), %rdi
.L2:
        testq        %rdi, %rdi
        jne         .L3
        cmpq        $100, %rax
        jne         .L5
        movq        $1, %rax
        ret
.L5:
        movq        $0, %rax
        ret
```

Write Y86-64 code that matches the given C and x86-64. You should mostly be able to translate instruction by instruction, but some x86-64 instructions will need to be implemented by multiple Y86-64 instructions.

Problem 3: Structs in Assembly (Based on practice problem 3.44)

Answer the following questions for each given struct, according to the x86-64 rules for structure layout and alignment: **(a)** What is the size of the struct in bytes? **(b)** What is the offset of the third field (always named `t`)? **(c)** What is the required alignment of the struct? **(d)** Is there a way to order the fields such that fewer bytes are wasted due to alignment? If so, give an order that has the smallest possible size.

```
struct P1 {char c; int i; char t; int j;};

struct P2 {long l; char *p; int t; short s;};

struct P3 {int i[3]; char c[3]; short t[2];};

struct P4 {char c; int i; double *t; short s; float f[2];}
```

Draw the layout of the struct P4 as a sequence of bytes in memory. Draw each byte labeled by the letter of the variable that the byte is part of, or write X for unused bytes. An example is given below:

```
struct P{int i; char c[2];};

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
  i   i   i   i   c   c   X   X
+---+---+---+---+---+---+---+---+
```

Notes on drawing the struct:

Make sure to include bytes that are left unused due to alignment. Also, notice how the variable c, an array, is only represented in the total number of bytes, not distinguishing between the elements within the array. Lastly, please use the variable names given in the struct above.

Problem 4: Memory Allocation

The following C program has many errors with memory allocation using `malloc()` and `free()`, which might prevent the code from compiling, or make it crash or have other undesirable behavior when run. Identify all these errors. For each error, briefly explain the issue and suggest a way to fix it. The code is divided in to four sections: A, B, C, and D. All of the dynamically allocated memory should be `free`d before beginning the next section. There may be more than one error in each section.

```
struct pet {
    char name[64];
    int species;
    int friendliness;
    float weight;
};

int main() {
// A
    int static_array[5] = {1, 2, 3, 4, 5};
    int *dynamic_array = static_array;
    *dynamic_array = 10;
    free(dynamic_array);

// B
    int static_array_2[5] = {1, 2, 3, 4, 5};
    int *dynamic_array_2 = (int *)malloc(5 * sizeof(int));
    for(int i = 0; i <= 5; i++) {
        dynamic_array_2[i] = static_array_2[5 - i];
    }
    free(dynamic_array_2);

// C
    struct pet max = {"Max", 1, 15, 60.5};
    struct pet *max_ptr = (struct pet*)malloc(sizeof(struct pet));
    *max_ptr = max;
    max_ptr->name = "Max";
    free(*max_ptr);

// D
    struct pet** my_pets = (struct pet**)malloc(10 * sizeof(struct pet));
    for(int i = 0; i < 10; i++) {
        my_pets[i] = (struct pet*)malloc(sizeof(struct pet));
    }

    free(*my_pets);
}
```