ICS Homework 4

Cao Shengcao

3.68

From the assembly code we can infer:

- `t` in `struct str2` has the offset of 8 bytes
- `u` in `struct str2` has the offset of 32 bytes
- `y` in `struct str1` has the offset of 184 bytes

Based on the data alignment rules, we can list the following inequalities:

$$\begin{cases} 176 < 4AB \le 184 \\ 4 < B \le 8 \\ 12 < 2A \le 20 \end{cases} \Rightarrow \begin{cases} 45 \le AB \le 46 \\ 5 \le B \le 8 \\ 7 \le A \le 10 \end{cases} \Rightarrow \begin{cases} A = 9 \\ B = 5 \end{cases}$$

Hence, the definition is

```
#define A 9
#define B 5
```

3.69

- A. From the assembly code we can infer:
  - `sizeof(a_struct)` should be 40 bytes
  - $CNT \times$ `sizeof(a_struct)` should be 280 bytes, so $CNT = 7$
- B. Similarly,
  - `%rax` stores `(void*)bp+40i`, `%rdx` stores `bp->a[i]->idx`
  - Both `ap->idx` and `ap->x[ap->idx]` are 8 byte data, so

    ```
    typedef struct
    {
        long idx;
        long x[4];
    } a_struct;
    ```

3.70

- A. The offset in bytes:

Member	Offset
`e1.p`	0
`e1.y`	8
`e2.x`	0
`e2.next`	8

- B. The size of this struct is 16 bytes.
- C. We can interpret the assembly code step by step:
  - Get `up->e1.y` or `up->e2.next` in `%rax`
  - Get `up->e2.next->e1.p` or `up->e2.next->e2.x` in `%rdx`
  - Get `*(up->e2.next->e1.p)` in `%rdx`
  - Get `*(up->e2.next->e1.p) - up->e2.next->e1.y` in `%rdx`
  - Move the value into `up->e2.x`

  So the function is

```
void proc(union ele *up)
{
    up->e2.x = *(up->e2.next->e1.p) - up->e2.next->e1.y;
}
```

  Some details:

  - There are many instances of indirect addressing in the assembly code. Be careful to distinguish them.
  - Don't miss `e1.` or `e2.`, because variables like `x` are not direct members of `union ele`.
  - `a->b` is a kind of "syntactic sugar" for `(*a).b`, used when we need to access a member with a pointer. Note that the operator `.` has a higher precedence than `*`, so we have to write `(*up).e2.x` for the expression. `up->e2.x` looks nicer.