Esse é o relatório da questão segunda da segunda avaliação da cadeira de Arquitetura e Organização de Computadores I - 01A - 2020.1, sob o professor Wagner. Sou Pedro Henrique Magalhães Botelho, de matrícula 471047, e venho por meio desse texto informar ao excelentíssimo meu progresso e pensamentos conforme o passar do tempo a respeito da questão primeira.

Segue abaixo o enunciado da questão:

> Questão 02: Escreva um programa que implementa uma máquina de estados finitos (Finite State Machine - FSM) capaz de reconhecer um endereço de e-mail.
>
> Seu programa irá receber um endereço de e-mail e escrever "Válido" ou "Invalido", de acordo com o caso.
>
> Para fins deste exercício, um endereço de e-mail deve começar com uma letra ter somente um caractere '@' e pelo menos um domínio. Não poderá conter números e assuma que somente caracteres minúsculos são válidos.
>
> OBS: nos casos de teste há um enter após a entrada.
>
> **Exemplo de Entrada:**
>
> professor@ufc.br
>
> **Exemplo de Saída:**
>
> Valido

A questão pede pra implementar uma máquina de estados finitos (**Finite State Machine - FSM**) capaz de reconhecer um endereço de e-mail.

Primeiramente é interessante desenhar o grafo para essa FSM. No grafo temos 5 estados, em que A é o estado final e E é o estado final.

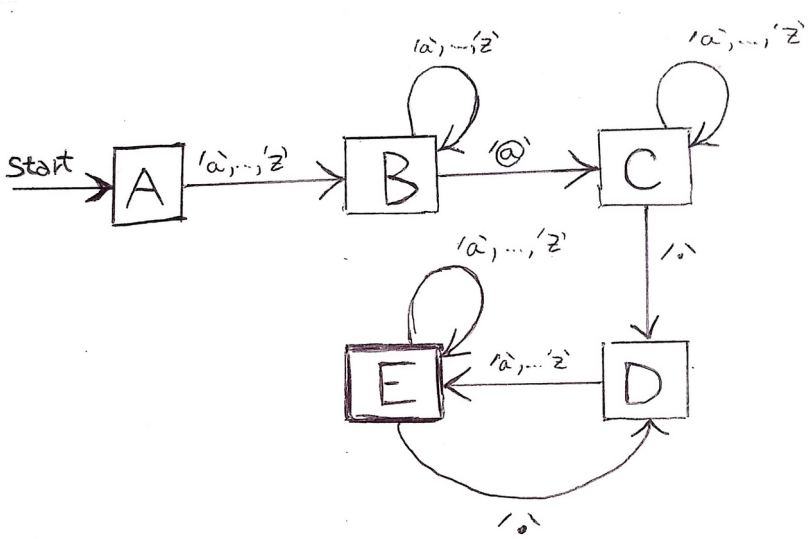

Figura 1: Grafo da FSM que reconhece um endereço de e-mail.

Vamos ver o que cada estado faz (lembrando que todos eles verificam pelo próximo caractere e se sua condição não for satisfeita é exibido uma mensagem de erro e o programa é encerrado):

- **Estado A**: Estado inicial, reconhece se o primeiro caractere inserido é uma letra minúscula.
- **Estado B:** Verifica se o caractere inserido é uma letra minúscula (se sim repete o próprio estado B) ou um arroba (passando para o estado C).
- **Estado C:** Verifica se o caractere inserido é uma letra minúscula (se sim repete o próprio estado C) ou um ponto (passando para o estado D).
- **Estado D**: Verifica se o caractere inserido é uma letra minúscula (se sim passa para o estado E).
- **Estado E**: Verifica se o caractere inserido é uma letra minúscula (se sim repete o próprio estado E) ou um ponto (passando para o estado D).

O estado D existe apenas para tratar do caractere ponto ('.'), para que não haja conflito e para não poder existir um domínio com dois pontos seguidos.

Por uma visão rápida do programa temos que os cinco estados estão no procedimento principal, e as ações estão em procedimentos devidamente documentados. O programa tem um **buffer** pra armazenar temporariamente o dado inserido, para depois ser passado a **AL**. Ainda uma posição de no máximo 50 bytes irá salvar o endereço de e-mail completo. Vamos ver a sintaxe de um dos estados (os cinco estados tem uma sintaxe semelhante, então vamos analisar apenas um):

```
1. STATE_C:
2. CALL GET_NEXT_INPUT
3. CALL IS_LETTER
4. JZ STATE_C
5. CALL IS_DOT_CHAR
6. JZ STATE_D
7. MOV EAX, invalidAddressMessage
8. CALL DISPLAY_MESSAGE
9. JMP end_main
```

O estado C, primeiramente, pega um caractere inserido, verifica se é uma letra ou um ponto. Se for uma letra o estado se repete e se for um ponto o estado D assume. Se não for nenhum é chamado o procedimento DISPLAY_MESSAGE e é impresso uma mensagem de erro.

Em suma, todos os estados seguem uma mesma linha. Vamos agora ver rapidamente cada um dos procedimentos.

- **GET_NEXT_INPUT:** pega um char digitado pelo usuário, coloca em AL e na string que guarda o endereço completo.
- **IS_LETTER:** verifica o char digitado (e em AL) é uma letra minúscula (entre 'a' e 'z') e retorna ZF como 1 se for.
- **IS_AT_CHAR:** verifica o char digitado (e em AL) é um arroba ('@') e retorna ZF como 1 se for.
- **IS_DOT_CHAR:** verifica o char digitado (e em AL) é um ponto ('.') e retorna ZF como 1 se for.
- **DISPLAY_MESSAGE:** imprime uma mensagem (o endereço deve estar em EAX) na tela.

Algumas preferências de implementação foram adotadas, como salvar o endereço final na memória. Se quiser imprimir para ver, só retirar o comentário das linhas 114 e 115. Outra foi usar **scanf** para recebe os dados do usuário. Originalmente, ao invés de scanf era usada a chamada de sistema **read**. Porém tive erros no verificador do moodle, mesmo em outros canto estando certo.

1. MOV EAX, 3
2. MOV EBX, 1
3. MOV ECX, inputBuffer
4. MOV EDX, 1
5. INT 0x80

Tive alguns erros, como por exemplo escrever "professor@ufc..br", ele dava o erro no segundo ponto e imprimia o "br" na linha de comando posterior, veja só os dois programas com essa única diferença rodando:

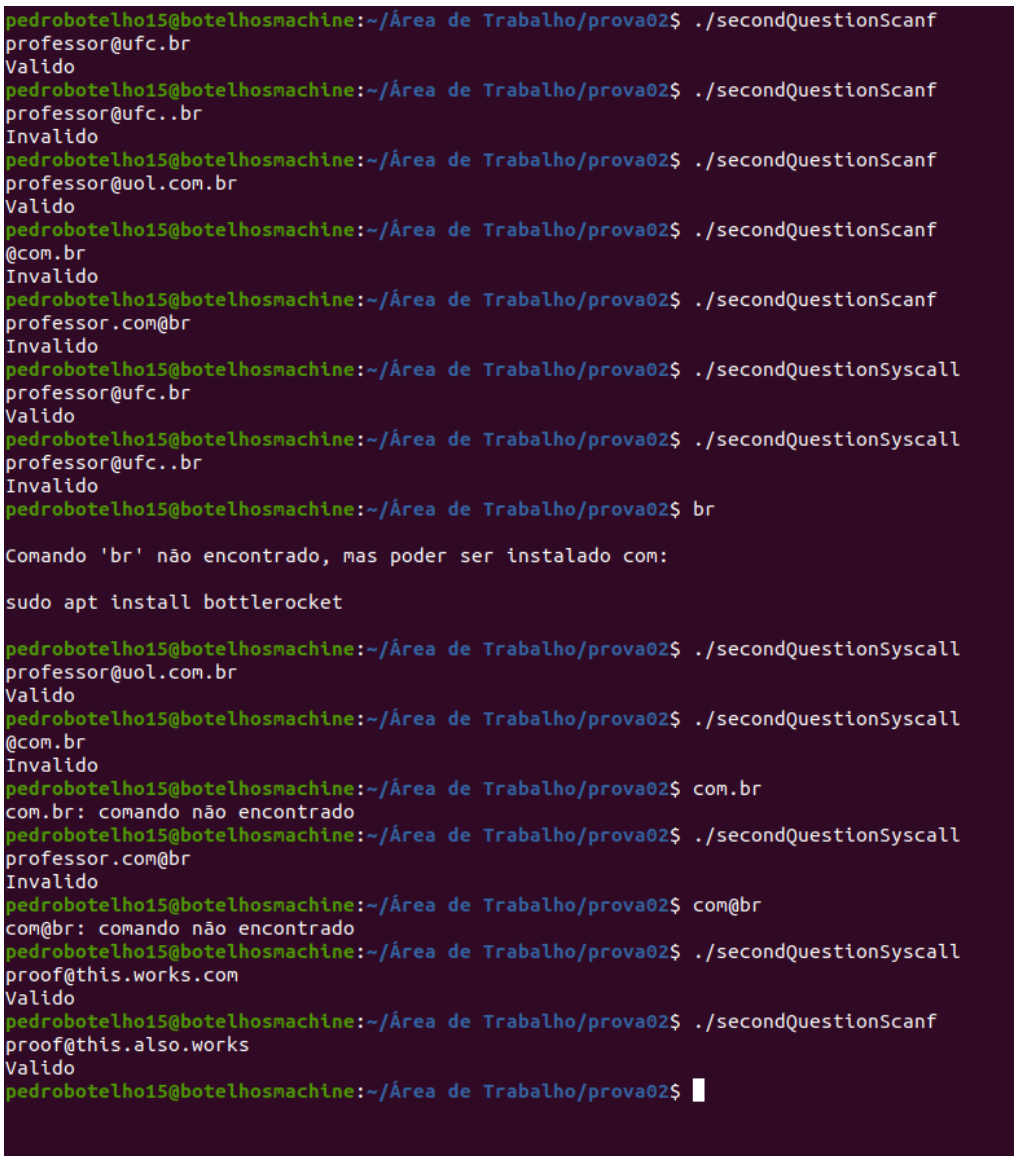

Figura 2: Diferenças entre SCANF e SYSCALL_READ

Então por isso acabei deixando apenas o **scanf**.

À parte disso não a nada a ser comentado.