

Explicativo sobre o programa praticas_09a.s

Este programa testa o uso da função malloc, da biblioteca libc. Tal função pode ser usada para alocar memória dinamicamente. A quantidade de memória (em bytes) deve ser passada no topo da pilha, antes de chamar a função com call. O endereço da região alocada é retornada no registrador %eax (endereço do primeiro byte).

No exemplo aqui apresentado, a função malloc é usada para alocar os bytes para um registro (conjunto de campos) acadêmico. Os campos são lidos, inseridos na região alocada, e depois lido e apresentado na tela.

O presente código aloca 80 bytes de memória para alocação de um registro contendo os seguintes campos de dados, em ordem:

| Nome | RA | Sexo | Curso | Prox |
|------|----|------|-------|------|
|------|----|------|-------|------|

1. Nome: 43 caracteres de tamanho + 1 caractere de final de string = 44 bytes
2. RA: número inteiro = 4 bytes
3. Sexo: 1 caractere 'M' ou 'F' = 4 bytes (para alinhamento de memória)
4. Curso: 23 caracteres de tamanho + 1 caractere de final de string = 24 bytes
5. Prox: endereço de outro registro para encadeamento = 4 bytes

Para gerar o executável, gere primeiro o objeto executando o seguinte comando:

```
as praticas_09a.s -o praticas_09a.o
```

e depois link dinamicamente com o seguinte comando:

```
ld praticas_09a.o -l c -dynamic-linker /lib/ld-linux.so.2 -o praticas_09a
```

O executável se chamara praticas_09a, sem extensão, e para executá-lo digite:

```
./praticas_09a
```

```
.section .data
```

```
# definicao das mensagens do programa
```

```
titulo:      .asciz      "\nPrograma que aloca a memoria e insere registros\n\n"
pedenome:    .asciz      "\nDigite o nome: "
pedera:      .asciz      "\nDigite o ra: "
pedesexo:    .asciz      "\nQual o sexo, <F>eminino ou <M>asculino?: "
pedecurso:   .asciz      "\nDigite o nome do curso: "

mostranome:  .asciz      "\nNome: %s"
mostrara:    .asciz      "\nRA: %d"
mostrasexo:  .asciz      "\nSexo: %c"
mostracurso: .asciz      "\nCurso: %s\n"

mostrapt:    .asciz      "\nptlista = %d\n"
```

```

formastr:    .asciz    "%s"
formach:     .asciz    "%c"
formanum:    .asciz    "%d"

pulalinha:   .asciz    "\n"

NULL:        .int     0    # representa posição nula da memoria

nalloc:      .int     80
ptlista1:    .int     0

```

```
.section .text
```

```

.globl      _start
_start:

                jmp      main

```

A funcao abaixo espera que o endereco inicial da memoria alocada esteja em %edi. Entao, ela le nome, RA, sexo e curso e coloca na respectiva memoria

```

le_dados:
    pushl     %edi        # endereco incial registro

    pushl     $pedenome
    call      printf
    addl      $4, %esp
    call      gets

    popl      %edi        # recupera %edi
    addl      $44, %edi    # avanca para o proximo campo
    pushl     %edi        # armazena na pilha

    pushl     $pedera
    call      printf
    addl      $4, %esp
    pushl     $formanum
    call      scanf
    addl      $4, %esp

    popl      %edi        # recupera %edi
    addl      $4, %edi    # avanca para o proximo campo
    pushl     %edi        # armazena na pilha

    pushl     $formach    # para remover o enter
    call      scanf
    addl      $4, %esp

    pushl     $pedesexo
    call      printf
    addl      $4, %esp
    pushl     $formach
    call      scanf
    addl      $4, %esp

```

```

    popl    %edi        # recupera %edi
    addl    $4, %edi    # avanca para o proximo campo
    pushl   %edi        # armazena na pilha

    pushl   $formach    # para remover o enter
    call    scanf
    addl    $4, %esp

    pushl   $pedecurso
    call    printf
    addl    $4, %esp

    call    gets        # pushl    $formastr
    #addl   $4, %esp

    popl    %edi        # recupera %edi
    addl    $24, %edi    # avanca para o proximo campo
    movl    $NULL, (%edi)

    subl    $76,%edi    # deixa %edi tal como estava no inicio

    RET

```

A funcao abaixo mostra os campos da memoria apontada por %edi, a saber: nome, RA, sexo e curso e coloca na memoria apontada %edi

mostra_dados:

```

    pushl   %edi        # endereco incial do registro

    pushl   $mostranome
    call    printf
    addl    $4, %esp

    popl    %edi        # recupera %edi
    addl    $44, %edi    # avanca para o proximo campo
    pushl   %edi        # armazena na pilha

    pushl   (%edi)
    pushl   $mostrara
    call    printf
    addl    $8, %esp

    popl    %edi        # recupera %edi
    addl    $4, %edi    # avanca para o proximo campo
    pushl   %edi        # armazena na pilha

    pushl   (%edi)
    pushl   $mostralsex
    call    printf
    addl    $8, %esp

    popl    %edi        # recupera %edi
    addl    $4, %edi    # avanca para o proximo campo
    pushl   %edi        # armazena na pilha

    pushl   $mostracurso
    call    printf
    addl    $4, %esp

```

```

        popl        %edi        # recupera %edi

        subl        $52,%edi    # deixa %edi tal como estava no inicio

        RET

main:

        pushl       $titulo
        call        printf

        movl        nalloc, %ecx
        pushl       %ecx
        call        malloc
        movl        %eax, ptlista1

        pushl       ptlista1
        pushl       $mostrapt
        call        printf

        addl        $16, %esp

        movl        ptlista1, %edi
        call        le_dados
        call        mostra_dados

fim:

        pushl       $0
        call        exit

```

DESAFIO: Ajuste o código para ler o RG depois do RA. O RG deve ser uma string. Ajuste o nalloc e o código. Aloque 2 registros e faça o primeiro apontar para o segundo. Imprima o 2 registros partindo do endereço inicial do primeiro.