AC2 - Exercício 11 - João Comini César de Andrade

Atividade 1

```
.data
prompt: .string "Digite um numero: "
res:    .string "Numero lido: "
newline:.string "\n"

.text
.globl main

main:
    # Chama a função
    call LeNumero
    mv s0, a0        # Salva o retorno em s0

    # Imprime mensagem
    li a0, 4
    la a1, res
    ecall

    # Imprime o número lido
    li a0, 1
    mv a1, s0
    ecall

    # Nova linha
    li a0, 4
    la a1, newline
    ecall

    # Encerra
    li a0, 10
```

```
    ecall

# --- Função LeNumero ---
# Retorna: a0 (inteiro lido)
LeNumero:
    # Imprime prompt (opcional, mas bom para usabilidade)
    li a0, 4
    la a1, prompt
    ecall

    # Lê inteiro (syscall 5)
    li a0, 5
    ecall
    ret
```

Atividade 2

```
.data
vetor:   .space 40        # Espaço para 10 inteiros
prompt: .string "Digite um numero para o vetor: "
msg:     .string "Vetor preenchido.\n"

.text
.globl main

main:
    la a0, vetor          # Endereço do vetor
    li a1, 5              # N = 5 elementos
    call LeVetor

    # Mensagem final
    li a0, 4
    la a1, msg
    ecall
```

```
    li a0, 10
    ecall

# --- Função LeVetor ---
# void LeVetor(int *v (a0), int N (a1))
LeVetor:
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s0, 4(sp)
    sw s1, 8(sp)

    mv s0, a0           # s0 = ponteiro vetor
    mv s1, a1           # s1 = contador N

loop_le_vetor:
    beqz s1, fim_le_vetor

    call LeNumero       # Chama função auxiliar
    sw a0, 0(s0)        # Guarda no vetor

    addi s0, s0, 4      # Próxima posição
    addi s1, s1, -1     # Decrementa contador
    j loop_le_vetor

fim_le_vetor:
    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 12
    ret

# --- Função LeNumero ---
```

```
LeNumero:
    li a0, 4
    la a1, prompt
    ecall
    li a0, 5
    ecall
    ret
```

Atividade 3

```
.data
vetor:  .space 40
prompt: .string "Digite valor: "
msg_med:.string "A media eh: "

.text
.globl main

main:
    # 1. Lê o vetor (vamos ler 4 números)
    la a0, vetor
    li a1, 4
    call LeVetor

    # 2. Calcula a média
    la a0, vetor
    li a1, 4
    call Media

    # 3. Imprime a média
    mv s0, a0       # Salva média
    li a0, 4
```

```
    la a1, msg_med
    ecall

    li a0, 1
    mv a1, s0
    ecall

    li a0, 10
    ecall

# --- Função Media ---
# int Media(int *v (a0), int N (a1))
Media:
    mv t0, a0       # t0 = vetor
    mv t1, a1       # t1 = N
    li t2, 0        # t2 = soma

loop_media:
    beqz t1, calc_div
    lw t3, 0(t0)
    add t2, t2, t3
    addi t0, t0, 4
    addi t1, t1, -1
    j loop_media

calc_div:
    div a0, t2, a1  # a0 = soma / N
    ret

# --- Função LeVetor ---
LeVetor:
    addi sp, sp, -12
    sw ra, 0(sp)
```

```
    sw s0, 4(sp)
    sw s1, 8(sp)
    mv s0, a0
    mv s1, a1
loop_lv:
    beqz s1, fim_lv
    call LeNumero
    sw a0, 0(s0)
    addi s0, s0, 4
    addi s1, s1, -1
    j loop_lv
fim_lv:
    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 12
    ret

# --- Função LeNumero ---
LeNumero:
    li a0, 4
    la a1, prompt
    ecall
    li a0, 5
    ecall
    ret
```

Atividade 4

```
.data
vetor: .space 400   # Espaço seguro
prompt: .string "Digite valor: "
msg_n:  .string "Quantos numeros vai digitar? "
```

```
msg_res:.string "Quantidade maiores que a media: "

.text
.globl main

main:
    # Pergunta N
    li a0, 4
    la a1, msg_n
    ecall
    li a0, 5
    ecall
    mv s0, a0       # s0 = N

    # Lê Vetor
    la a0, vetor
    mv a1, s0
    call LeVetor

    # Calcula Média
    la a0, vetor
    mv a1, s0
    call Media
    mv s1, a0       # s1 = Média

    # Conta maiores que a média
    la s2, vetor    # s2 = ponteiro
    mv s3, s0       # s3 = contador
    li s4, 0        # s4 = resposta (qtd)

loop_conta:
    beqz s3, fim_conta
    lw t0, 0(s2)    # carrega valor
```

```
    ble t0, s1, proximo # se valor <= media, ignora
    addi s4, s4, 1      # se maior, conta++

proximo:
    addi s2, s2, 4
    addi s3, s3, -1
    j loop_conta

fim_conta:
    li a0, 4
    la a1, msg_res
    ecall
    li a0, 1
    mv a1, s4
    ecall
    li a0, 10
    ecall

# (Copiar funções Media, LeVetor, LeNumero das atividades anteriores
aqui.
# Para economizar espaço visual, assuma que estão aqui idênticas à
Atividade 3)
# --- Função Media ---
Media:
    mv t0, a0       # t0 = vetor
    mv t1, a1       # t1 = N
    li t2, 0        # t2 = soma
loop_media:
    beqz t1, calc_div
    lw t3, 0(t0)
    add t2, t2, t3
    addi t0, t0, 4
```

```
    addi t1, t1, -1
    j loop_media
calc_div:
    div a0, t2, a1  # a0 = soma / N
    ret

# --- Função LeVetor ---
LeVetor:
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s0, 4(sp)
    sw s1, 8(sp)
    mv s0, a0
    mv s1, a1
loop_lv:
    beqz s1, fim_lv
    call LeNumero
    sw a0, 0(s0)
    addi s0, s0, 4
    addi s1, s1, -1
    j loop_lv
fim_lv:
    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 12
    ret

# --- Função LeNumero ---
LeNumero:
    li a0, 4
    la a1, prompt
    ecall
```

```
    li a0, 5
    ecall
    ret
```

Atividade 5

O programa imprime "Digite um numero".
Você digita 3 e aperta Enter.
O que acontece (O Bug): O programa trava na leitura. Ele não avança para ler os próximos números do vetor, nem calcula a média. Ele continua esperando você digitar infinitamente. Qualquer coisa que você digite a seguir apenas é "engolida" pelo programa, mas ele nunca sai da primeira função de leitura.
Aconteceu o que você esperava? Não. O esperado (baseado na `main`) era que ele lesse o tamanho N (3), depois lesse 3 números, calculasse a média e imprimisse os resultados. O programa falha em cumprir sua função básica.

Atividade 6

```
# Programa de teste da correção
.data
buffer: .space 20
msg: .string "Digite algo e aperte Enter: "

.text
.globl main
main:
    li a0, 4
    la a1, msg
    ecall

    la a0, buffer
    call LeString

    li a0, 4
    la a1, buffer
    ecall

    li a0, 10
```

```
    ecall

# --- LeString Corrigida (Atividade 6) ---
LeString:
    addi sp, sp, -8
    sw s0, 4(sp)
    sw ra, 0(sp)

    mv s0, a0           # s0 = endereço do buffer

    # Habilita teclado
    li a0, 0x130
    ecall

leitura_loop:
    li a0, 0x131        # Verifica se tem char
    ecall

    beqz a0, fim_le_str # Se 0, acabou entrada (raro no simulador
manual)
    li t0, 1
    beq a0, t0, leitura_loop # Se 1, nada digitado, tenta de novo

    # Se chegou aqui, a0=2 e o char está em a1
    li t1, 10           # ASCII do Enter (\n)
    beq a1, t1, fim_le_str # CORREÇÃO: Se for Enter, sai do loop!

    sb a1, 0(s0)        # Salva char
    addi s0, s0, 1      # Avança ponteiro
    j leitura_loop

fim_le_str:
    sb zero, 0(s0)      # Adiciona \0 no final
```

```
    lw ra, 0(sp)
    lw s0, 4(sp)
    addi sp, sp, 8
    ret
```

Atividade 7

```
.data
buffer_temp: .space 100
msg: .string "Digite uma string: "

.text
.globl main
main:
    call LeStringDinamica
    mv s0, a0           # Salva o ponteiro retornado (heap)

    # Imprime a string que está no Heap
    li a0, 4
    mv a1, s0
    ecall

    li a0, 10
    ecall

# --- LeStringDinamica ---
LeStringDinamica:
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s0, 4(sp)
    sw s1, 8(sp)

    # 1. Lê para buffer estático
```

```
    li a0, 4
    la a1, msg
    ecall

    la a0, buffer_temp
    call LeStringSimples # Usa a função corrigida da ativ anterior

    # 2. Calcula tamanho (strlen)
    la a0, buffer_temp
    call strlen
    mv s0, a0            # s0 = tamanho

    # 3. Aloca memória (sbrk)
    addi a0, s0, 1       # tamanho + 1 (para o \0)
    li a7, 9             # sbrk (malloc)
    mv a1, a0
    ecall
    mv s1, a0            # s1 = endereço da nova memória

    # 4. Copia string (strcpy)
    mv a0, s1            # destino (heap)
    la a1, buffer_temp   # origem (static)
    call strcpy

    mv a0, s1            # Retorna o endereço do heap

    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 12
    ret

# --- Funções Auxiliares Necessárias ---
```

```
LeStringSimples: # Versão compacta da LeString corrigida
    mv t0, a0
    li a7, 0x130
    ecall
l_loop:
    li a7, 0x131
    ecall
    li t1, 1
    beq a7, t1, l_loop
    li t2, 10
    beq a1, t2, l_end
    sb a1, 0(t0)
    addi t0, t0, 1
    j l_loop
l_end:
    sb zero, 0(t0)
    ret

strlen:
    mv t0, a0
len_loop:
    lbu t1, 0(t0)
    beqz t1, len_end
    addi t0, t0, 1
    j len_loop
len_end:
    sub a0, t0, a0
    ret

strcpy:
    mv t0, a0
cpy_loop:
    lbu t1, 0(a1)
```

```
    sb t1, 0(a0)
    beqz t1, cpy_end
    addi a0, a0, 1
    addi a1, a1, 1
    j cpy_loop
cpy_end:
    mv a0, t0
    ret
```

Atividade 8

```
.data
buffer_temp: .space 100
prompt_nm: .string "Nome: "
prompt_id: .string "Idade: "

.text
.globl main
main:
    # Cria uma pessoa
    call LePessoa

    # Imprime a pessoa criada
    call ImprimePessoa

    li a0, 10
    ecall

# --- LePessoa (Dinâmica) ---
LePessoa:
    addi sp, sp, -16
    sw ra, 0(sp)
    sw s0, 4(sp)
```

```
    # 1. Lê Nome Dinâmico
    call LeStringDinamica
    mv s0, a0               # s0 = ponteiro do nome

    # 2. Aloca Struct (12 bytes)
    li a7, 9
    li a1, 12
    ecall
    mv t0, a0               # t0 = endereço da struct

    # 3. Preenche Struct
    sw s0, 0(t0)            # guarda ponteiro do nome

    # Lê Idade
    li a0, 4
    la a1, prompt_id
    ecall
    li a0, 5
    ecall
    sw a0, 4(t0)            # guarda idade

    sw zero, 8(t0)          # prox = NULL

    mv a0, t0               # Retorna struct

    lw ra, 0(sp)
    lw s0, 4(sp)
    addi sp, sp, 16
    ret
# --- Funções Auxiliares (LeStringDinamica e dependencias) ---
# (LeStringDinamica, LeStringSimples, strlen, strcpy)
LeStringDinamica:
```

```
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s0, 4(sp)
    sw s1, 8(sp)
    li a0, 4
    la a1, prompt_nm
    ecall
    la a0, buffer_temp
    call LeStringSimples
    la a0, buffer_temp
    call strlen
    mv s0, a0
    addi a0, s0, 1
    li a7, 9
    mv a1, a0
    ecall
    mv s1, a0
    mv a0, s1
    la a1, buffer_temp
    call strcpy
    mv a0, s1
    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 12
    ret
LeStringSimples:
    mv t0, a0
    li a7, 0x130
    ecall
l_ls: li a7, 0x131
    ecall
    li t1, 1
```

```
    beq a7, t1, l_ls
    li t2, 10
    beq a1, t2, e_ls
    sb a1, 0(t0)
    addi t0, t0, 1
    j l_ls
e_ls: sb zero, 0(t0)
    ret
strlen:
    mv t0, a0
l_len: lbu t1, 0(t0)
    beqz t1, e_len
    addi t0, t0, 1
    j l_len
e_len: sub a0, t0, a0
    ret
strcpy:
    mv t0, a0
l_cpy: lbu t1, 0(a1)
    sb t1, 0(a0)
    beqz t1, e_cpy
    addi a0, a0, 1
    addi a1, a1, 1
    j l_cpy
e_cpy: mv a0, t0
    ret
```

Atividade 9

```
.data
buffer_temp: .space 100
prompt_nm: .string "Nome: "
```

```
prompt_id: .string "Idade: "

.text
.globl main
main:
    # Cria uma pessoa
    call LePessoa

    # Imprime a pessoa criada
    call ImprimePessoa

    li a0, 10
    ecall

# --- LePessoa (Dinâmica) ---
LePessoa:
    addi sp, sp, -16
    sw ra, 0(sp)
    sw s0, 4(sp)

    # 1. Lê Nome Dinâmico
    call LeStringDinamica
    mv s0, a0              # s0 = ponteiro do nome

    # 2. Aloca Struct (12 bytes)
    li a7, 9
    li a1, 12
    ecall
    mv t0, a0              # t0 = endereço da struct

    # 3. Preenche Struct
    sw s0, 0(t0)           # guarda ponteiro do nome
```

```
    # Lê Idade
    li a0, 4
    la a1, prompt_id
    ecall
    li a0, 5
    ecall
    sw a0, 4(t0)          # guarda idade

    sw zero, 8(t0)        # prox = NULL

    mv a0, t0             # Retorna struct

    lw ra, 0(sp)
    lw s0, 4(sp)
    addi sp, sp, 16
    ret

# --- ImprimePessoa ---
ImprimePessoa:
    addi sp, sp, -4
    sw ra, 0(sp)
    mv t0, a0             # t0 = struct

    # Imprime Nome
    lw a0, 0(t0)          # carrega ptr nome
    li a7, 4
    ecall

    # Espaço
    li a0, 11
    li a1, 32
    ecall
```

```
    # Imprime Idade
    lw a0, 4(t0)          # carrega idade
    li a7, 1
    ecall

    # Newline
    li a0, 11
    li a1, 10
    ecall

    lw ra, 0(sp)
    addi sp, sp, 4
    ret

# --- Funções Auxiliares (LeStringDinamica e dependencias) ---
# (LeStringDinamica, LeStringSimples, strlen, strcpy)
LeStringDinamica:
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s0, 4(sp)
    sw s1, 8(sp)
    li a0, 4
    la a1, prompt_nm
    ecall
    la a0, buffer_temp
    call LeStringSimples
    la a0, buffer_temp
    call strlen
    mv s0, a0
    addi a0, s0, 1
    li a7, 9
    mv a1, a0
    ecall
```

```
    mv s1, a0
    mv a0, s1
    la a1, buffer_temp
    call strcpy
    mv a0, s1
    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 12
    ret
LeStringSimples:
    mv t0, a0
    li a7, 0x130
    ecall
l_ls: li a7, 0x131
    ecall
    li t1, 1
    beq a7, t1, l_ls
    li t2, 10
    beq a1, t2, e_ls
    sb a1, 0(t0)
    addi t0, t0, 1
    j l_ls
e_ls: sb zero, 0(t0)
    ret
strlen:
    mv t0, a0
l_len: lbu t1, 0(t0)
    beqz t1, e_len
    addi t0, t0, 1
    j l_len
e_len: sub a0, t0, a0
    ret
```

```
strcpy:
    mv t0, a0
l_cpy: lbu t1, 0(a1)
    sb t1, 0(a0)
    beqz t1, e_cpy
    addi a0, a0, 1
    addi a1, a1, 1
    j l_cpy
e_cpy: mv a0, t0
    ret
```

Atividade 10

```
.data
# Buffer temporário para leitura inicial da string (antes de saber o
tamanho exato)
buffer_temp: .space 100
prompt_nome: .string "Digite o Nome: "
prompt_idade:.string "Digite a Idade: "
msg_saida:   .string "\n--- Dados da Pessoa (Struct Dinamica) ---\n"
txt_nome:    .string "Nome: "
txt_idade:   .string "Idade: "

.text
.globl main

# ================================================================
# MAIN: Testa a criação e impressão da Pessoa
# ================================================================
main:
    # 1. Chama LePessoa (Retorna o endereço da struct em a0)
    call LePessoa
    mv s0, a0           # Salva o ponteiro da struct em s0
```

```
    # 2. Imprime cabeçalho
    li a0, 4
    la a1, msg_saida
    ecall

    # 3. Chama ImprimePessoa passando a struct
    mv a0, s0
    call ImprimePessoa

    # 4. Encerra o programa
    li a0, 10
    ecall

# ========================================================================
# ATIVIDADE 8/10: LePessoa
# Aloca struct dinâmica (12 bytes) e usa nome dinâmico
# Retorno: a0 = endereço da struct alocada
# ========================================================================
LePessoa:
    addi sp, sp, -16
    sw ra, 0(sp)
    sw s0, 4(sp)        # Vai guardar o ponteiro do nome
    sw s1, 8(sp)        # Vai guardar o ponteiro da struct

    # 1. Ler o Nome Dinamicamente (Função da Atividade 7)
    #    Isso retorna um ponteiro para o Heap em a0
    call LeString
    mv s0, a0           # s0 = char *nome

    # 2. Alocar memória para a STRUCT Pessoa
    #    Tamanho: 4 (ptr nome) + 4 (int idade) + 4 (ptr prox) = 12
bytes
```

```
    li a7, 9            # sbrk (malloc)
    li a1, 12           # tamanho
    ecall
    mv s1, a0           # s1 = endereço da struct (Pessoa*)

    # 3. Preencher a Struct
    sw s0, 0(s1)        # pessoa->nome = s0 (ponteiro do nome)

    # 4. Ler a Idade
    li a0, 4
    la a1, prompt_idade
    ecall
    li a0, 5            # read_int
    ecall
    sw a0, 4(s1)        # pessoa->idade = a0

    # 5. Inicializar prox com NULL
    sw zero, 8(s1)      # pessoa->prox = NULL

    # Retorna o endereço da struct
    mv a0, s1

    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 16
    ret

# ================================================================
# ATIVIDADE 9: ImprimePessoa
# Recebe: a0 = endereço da struct Pessoa
# ================================================================
ImprimePessoa:
```

```
    addi sp, sp, -4
    sw ra, 0(sp)

    mv t0, a0           # t0 = endereço da struct

    # Imprime Label "Nome: "
    li a0, 4
    la a1, txt_nome
    ecall

    # Imprime o CONTEÚDO do nome
    # Atenção: 0(t0) contém o ENDEREÇO da string, não a string em si.
    lw a0, 0(t0)        # Carrega o ponteiro char*
    li a7, 4            # print_string
    ecall

    # Nova linha
    li a0, 11
    li a1, 10
    ecall

    # Imprime Label "Idade: "
    li a0, 4
    la a1, txt_idade
    ecall

    # Imprime a Idade
    lw a0, 4(t0)        # Carrega o int idade
    li a7, 1            # print_int
    ecall

    # Nova linha final
    li a0, 11
```

```
    li a1, 10
    ecall

    lw ra, 0(sp)
    addi sp, sp, 4
    ret

# ================================================================
# ATIVIDADE 7: LeString (Alocação Dinâmica)
# Lê string do teclado -> Buffer temp -> Aloca Heap -> Copia
# Retorno: a0 = endereço da string no Heap
# ================================================================
LeString:
    addi sp, sp, -16
    sw ra, 0(sp)
    sw s0, 4(sp)          # Tamanho
    sw s1, 8(sp)          # Ponteiro Heap

    # 1. Prompt
    li a0, 4
    la a1, prompt_nome
    ecall

    # 2. Ler para buffer estático temporário
    la a0, buffer_temp
    call LeStringSimples # Lê via polling

    # 3. Calcular tamanho exato da string lida
    la a0, buffer_temp
    call strlen
    mv s0, a0             # s0 = tamanho

    # 4. Alocar memória no Heap (malloc)
```

```
    addi a0, s0, 1        # Tamanho + 1 (para o \0)
    li a7, 9              # sbrk
    mv a1, a0
    ecall
    mv s1, a0             # s1 = endereço da nova memória

    # 5. Copiar do buffer estático para o Heap
    mv a0, s1             # Destino
    la a1, buffer_temp    # Origem
    call strcpy

    # Retorna endereço do Heap
    mv a0, s1

    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 16
    ret

# ================================================================
# FUNÇÕES AUXILIARES (Strings e Polling)
# ================================================================

# Leitura caractere por caractere (Polling)
# a0 = buffer destino
LeStringSimples:
    mv t0, a0
    # Habilita teclado
    li a7, 0x130
    ecall
loop_ls:
    li a7, 0x131          # Check teclado
```

```
    ecall
    li t1, 1
    beq a7, t1, loop_ls # Se nada digitado, repete
    beqz a7, fim_ls     # Se erro/eof, sai

    # Verifica Enter (\n = 10)
    li t2, 10
    beq a1, t2, fim_ls

    sb a1, 0(t0)        # Salva char
    addi t0, t0, 1
    j loop_ls

fim_ls:
    sb zero, 0(t0)      # Add \0
    ret

# strlen(char *s) -> a0 = tamanho
strlen:
    mv t0, a0
loop_len:
    lbu t1, 0(t0)
    beqz t1, fim_len
    addi t0, t0, 1
    j loop_len
fim_len:
    sub a0, t0, a0
    ret

# strcpy(char *dest, char *src) -> copia src para dest
strcpy:
    mv t0, a0           # Salva inicio dest
loop_cpy:
```

```
    lbu t1, 0(a1)
    sb t1, 0(a0)
    beqz t1, fim_cpy
    addi a0, a0, 1
    addi a1, a1, 1
    j loop_cpy
fim_cpy:
    mv a0, t0
    ret
```

Atividade 11

```
.data
buffer_temp: .space 100
prompt_nm: .string "Nome (Enter vazio para parar): "
prompt_id: .string "Idade: "
msg_lst:   .string "--- Lista ---\n"

.text
.globl main
main:
    li s0, 0        # s0 = HEAD da lista
    li s1, 0        # s1 = ULTIMO no inserido

loop_main:
    call LePessoa   # Tenta ler
    beqz a0, fim_le # Se retornou NULL, para

    # Inserção na Lista
    bnez s0, insere_fim
    mv s0, a0       # Primeiro nó
    mv s1, a0
    j loop_main
```

```
insere_fim:
    sw a0, 8(s1)     # ultimo->prox = novo
    mv s1, a0        # ultimo = novo
    j loop_main

fim_le:
    # Imprime Lista
    li a0, 4
    la a1, msg_lst
    ecall

    mv s2, s0        # s2 = atual
loop_prt:
    beqz s2, fim_pgm
    mv a0, s2
    call ImprimePessoa
    lw s2, 8(s2)     # atual = atual->prox
    j loop_prt

fim_pgm:
    li a0, 10
    ecall

# --- LePessoa Modificada para retornar NULL se nome vazio ---
LePessoa:
    addi sp, sp, -16
    sw ra, 0(sp)
    sw s0, 4(sp)

    # 1. Lê Nome
    call LeStringDinamica
```

```
    # Check se vazio (primeiro char é 0)
    lb t0, 0(a0)
    bnez t0, continua_le
    li a0, 0        # Retorna NULL
    j sai_lepessoa

continua_le:
    mv s0, a0       # s0 = nome
    # 2. Aloca Struct
    li a7, 9
    li a1, 12
    ecall
    mv t0, a0
    sw s0, 0(t0)

    # 3. Lê Idade
    li a0, 4
    la a1, prompt_id
    ecall
    li a0, 5
    ecall
    sw a0, 4(t0)
    sw zero, 8(t0)
    mv a0, t0

sai_lepessoa:
    lw ra, 0(sp)
    lw s0, 4(sp)
    addi sp, sp, 16
    ret
```

Atividade 12

```
.data
heap_head: .word 0 # Cabeça da lista de blocos livres/ocupados

.text
.globl main
main:
    # Teste simples do malloc
    li a0, 20       # Quero 20 bytes
    call malloc
    mv s0, a0       # s0 = ponteiro dados 1

    li a0, 10       # Quero 10 bytes
    call malloc
    mv s1, a0       # s1 = ponteiro dados 2

    # Teste do free
    mv a0, s0
    call free       # Libera o primeiro bloco

    # Reuso (deve pegar o bloco liberado se couber)
    li a0, 15
    call malloc     # Deve reusar o espaço de s0

    li a0, 10
    ecall

# --- malloc(int size) ---
malloc:
    # Alinha size p/ 4
    addi a0, a0, 3
    andi a0, a0, -4
```

```
    la t0, heap_head
    lw t1, 0(t0)     # t1 = primeiro bloco

search:
    beqz t1, create
    lw t2, 4(t1)     # ocupado?
    bnez t2, next
    lw t3, 0(t1)     # tamanho
    blt t3, a0, next # cabe?

    # Reusa bloco
    li t2, 1
    sw t2, 4(t1)
    addi a0, t1, 16 # Pula header (16 bytes)
    ret

next:
    lw t1, 12(t1)    # prox
    j search

create:
    mv t4, a0        # salva size
    addi a0, a0, 16 # size + header
    li a7, 9
    mv a1, a0
    ecall

    sw t4, 0(a0)     # header.tamanho
    li t2, 1
    sw t2, 4(a0)     # header.ocupado
    sw zero, 12(a0) # header.prox (simplificado, não encadeia no
exemplo basico)
```

```
    # Nota: Em um malloc real, aqui deveriamos atualizar o ->prox do
ultimo bloco
    # para apontar para este novo a0.

    addi a0, a0, 16 # retorna dados
    ret

# --- free(void *ptr) ---
free:
    addi a0, a0, -16
    sw zero, 4(a0)  # ocupado = 0
    ret
```