Aula Teórica 13 (guião)

Semana de 8 a 12 de Dezembro de 2025

José Carlos Ramalho

Sinopsis:

- Geração de código VM para:
    1. Arrays
    2. Funções
- Exemplos desenvolvidos:
    1. Ler 5 inteiros para um array e imprimir esse array;
    2. Alocar uma posição de memória, armazenar um valor e aceder-lhe;
    3. Alocando dois blocos...
    4. Ler n; ler n inteiros para um array dinâmico e imprimir esse array;
    5. Chamada a uma função;
    6. Chamada a função com 1 parâmetro;
    7. Função para ler uma lista de elementos;
    8. Função recursiva: fatorial.

Arrays

- A seguir apresenta-se uma série de exemplo de dificuldade crescente focando a geração de código envolvendo a manipulação de arrays;
- Fazer uma introdução aos arrays em memória:
    1. Endereço de base;
    2. Índices e offsets.

Exemplo 1: Ler 5 inteiros para um array e imprimir esse array

- Fluxo de operações na leitura - `a[i] = read()`
    1. Colocar o endereço de base na stack;
    2. Colocar o offset na stack;
    3. Fazer a leitura;
    4. Guardar: `storen`.

```
program ReadAndPrintArray;

var
  nums: array[1..5] of Integer;
  i: Integer;
```

```
begin
  writeln('Introduza 5 valores inteiros:');
  for i := 1 to 5 do
  begin
    write('Introduza o valor  ', i, ': ');
    readln(nums[i]);
  end;

  writeln;
  writeln('Acabou de introduzir:');
  for i := 1 to 5 do
  begin
    writeln(nums[i]);
  end;
end.
```

Geração de código: declaração de variáveis

Pascal:

```
var
  nums: array[1..5] of Integer;
  i: Integer;
```

VM:

```
        pushi 0  // i = 0
        pushn 5  // array[5] = {0}
```

Geração de código: nums[i] = read(...)

```
    write('Introduza o valor  ', i, ': ');
    readln(nums[i]);
```

VM:

```
        pushgp
        pushi 1
        padd      // calcula endr base de a
        pushg 0   // calcula indice (i): a[i]
        pushs "Introduza o valor "
        writes
        pushg 0
```

```
        pushi 1
        add
        writei
        pushs ": "
        writes      // write('Introduza o valor  ', i, ': ');
        read
        atoi
        storen // a[i] = read()
```

Geração de código: writeln(nums[i]);

```
    writeln(nums[i]);
```

VM:

```
        pushgp
        pushi 1
        padd
        pushg 0
        loadn
        writei    // print a[i]
        writeln
```

Geração de código: o programa completo

VM:

```
        start

        pushi 0  // i = 0
        pushn 5  // array[5] = {0}

        pushs "Introduza 5 valores inteiros:"
        writes
        writeln     // writeln('Introduza 5 valores inteiros:');
ciclo:
        pushgp
        pushi 1
        padd      // calcula endr base de a
        pushg 0   // calcula indice (i): a[i]
        pushs "Introduza o valor "
        writes
        pushg 0
        pushi 1
        add
        writei
```

```
        pushs ": "
        writes      // write('Introduza o valor  ', i, ': ');
        read
        atoi
        storen // a[i] = read()

        pushg 0 // i = i + 1
        pushi 1
        add
        storeg 0

        pushg 0 // Foram feitas 5 leituras?
        pushi 5
        equal   // i == 5
        jz ciclo
        writeln

        // Escreve a lista lida
        pushi 0
        storeg 0  // i = 0
        pushs "Acabou de introduzir: "
        writes
        writeln

ciclo2:
        pushgp
        pushi 1
        padd
        pushg 0
        loadn
        writei    // print a[i]
        writeln

        pushg 0 // i = i + 1
        pushi 1
        add
        storeg 0

        pushg 0 // Foram feitas 5 leituras?
        pushi 5
        equal   // i == 5
        jz ciclo2

        stop
```

Exemplo 2: Alocar uma posição de memória, armazenar um valor e aceder-lhe

- Quando se aloca espaço na Heap, o endereço de base desta é manipulado com `pushst offset`;
- Se tiver mais do que uma estrutura na Heap tenho de indicar o offset relativamente ao endereço de base.

```
start
    pushi 2     // Reservo espaço na Heap para 2 inteiros
    allocn

    pushst 0
    pushi 73
    store 0 // a[0] = 73

    pushst 0
    pushi 55
    store 1 // a[1] = 55

    pushst 0
    load 0
    writei

    pushst 0
    load 1
    writei
stop
```

Exemplo 3: Alocando dois blocos...

```
start
    alloc 2
    alloc 5

    pushst 0
    pushi 73
    store 0 // a[0] = 73

    pushst 0
    pushi 55
    store 1 // a[1] = 55

    pushst 1
    pushi 23
    store 0

    pushst 0
    load 0
    writei

    pushst 0
    load 1
    writei

    pushst 1
    load 0
```

```
        writei

    stop
```

Exemplo 4: Ler n; ler n inteiros para um array dinâmico e imprimir esse array

- O Pascal standard ou ISO Pascal não inclui arrays dinâmicos;
- O Free Pascal ou a extensão Delphi ao Pascal standard incluem a função `SetLength()` e a sintaxe declarativa `array of Integer` que permitem fazê-lo;
- Exemplo em Free Pascal:

```
program ReadNIntegers;

var
  nums: array of Integer;
  N, i: Integer;

begin
  write('Quantos elementos terá a sua lista? ');
  readln(N);

  setlength(nums, N);

  writeln('Introduza ', N, ' valores inteiros:');
  for i := 0 to N - 1 do
  begin
    write('Introduza o valor ', i + 1, ': ');
    readln(nums[i]);
  end;

  writeln;
  writeln('Foram armazenados:');
  for i := 0 to N - 1 do
    writeln(nums[i]);
end.
```

- Em Pascal standard:

```
program ReadNIntegers;

const
  MaxSize = 1000;
var
  nums: array[1..MaxSize] of Integer;
  N, i: Integer;

begin
  write('Quantos elementos terá a sua lista? ');
```

```
  readln(N);

  if (N < 1) or (N > MaxSize) then
    writeln('Erro: N tem de estar entre 1 e ', MaxSize)
  else
    begin
        writeln('Introduza ', N, ' valores inteiros:');
        for i := 0 to N - 1 do
            begin
                write('Introduza o valor ', i + 1, ': ');
                readln(nums[i]);
            end;

        writeln;
        writeln('Foram armazenados:');
        for i := 0 to N - 1 do
            writeln(nums[i]);
    end;
end.
```

Exemplo 5: Chamada a uma função

- Fluxo operacional de uma chamada de uma função:
  1. Colocar o endereço da label na stack: `pusha label`;
  2. Fazer a chamada: `call`;
  3. No fim da função retornar ao programa principal: `return`.

```
start
    pushs "Função simples que escreve 77 no monitor: "
    writes
    pusha f1
    call
    writeln
    pushs "That's all folks!"
    writes
stop

f1:
    pushi 77
    writei
    return
```

Exemplo 6: Chamada a função com 1 parâmetro

- Fluxo operacional de uma chamada de uma função com parâmetros e a devolver um resultado:
  1. Reservar espaço na stack para o resultado, no caso de um valor inteiro: `pushi 0`;
  2. Colocar os parâmetros na stack: `push? param`;

3. Colocar o endereço da label na stack: `pusha label`;
4. Fazer a chamada: `call`;
5. Aceder ao parâmetros usando o `frame pointer`: `pushfl x`;
6. Colocar o resultado na stack usando o `frame pointer`: `storel -(n+1)`, em que `n`é o número de parâmetros;
7. No fim da função retornar ao programa principal: `return`;
8. Limpar os parâmetros: `pop n`, em que `n` é o número de parâmetros.

```
start
    pushs "Função simples que calcula o dobro de 67: "
    writes
    pushi 0    // espaço para o resultado
    pushi 67   // parâmetro
    pusha dobro
    call
    pop 1
    writei
    writeln
    pushs "That's all folks!"
    writes
stop

dobro:
    pushl  -1   // get arg1
    pushi   2
    mul
    storel -2
    return
```

Exemplo 7: função para ler uma lista de elementos

```
start
    pushn 10    // reserva espaço para uma lista de 10 elementos

    pushs "Função que lê uma lista: "
    writes
    pushgp      // param 1: endereço base da lista
    pushi 5     // param 2: número de elementos a ler
    pusha leLista
    call

    pushgp      // param 1: endereço base da lista
    pushi 5     // param 2: número de elementos a escrever
    pusha escreveLista
    call

    writeln
    pushs "That's all folks!"
```

```
    writes
stop

leLista:
        pushi  0   // variável de controlo do número de elementos a ler
        storel 0

        pushs "Introduza 5 valores inteiros:"
        writes
        writeln     // writeln('Introduza 5 valores inteiros:');
    ciclo:
        pushl -2    // endereço base da lista

        pushl 0   // calcula indice (i): a[i]
        pushs "Introduza o valor "
        writes
        pushl 0
        pushi 1
        add
        writei
        pushs ": "
        writes      // write('Introduza o valor  ', i, ': ');
        read
        atoi
        storen // a[i] = read()

        pushl 0 // i = i + 1
        pushi 1
        add
        storel 0

        pushl 0 // Foram feitas 5 leituras?
        pushi 5
        equal   // i == 5
        jz ciclo
    return

escreveLista:
        pushi  0   // variável de controlo do número de elementos a ler
        storel 0
    ciclo2:
        pushl -2    // endereço base da lista
        pushl 0     // indice (i): a[i]
        loadn
        writei
        writeln

        pushl 0 // i = i + 1
        pushi 1
        add
        storel 0

        pushl 0 // Foram feitas 5 leituras?
        pushi 5
```

```
        equal   // i == 5
        jz ciclo2
    return
```

Desafio: Ler N; ler uma lista de N elementos; validar N

Exemplo 8: Função recursiva - fatorial.

- Num função recursiva, é preciso ter o cuidado de limpar os parâmetros da stack com `pop n`, em que `n` é o número de parâmetros.

```
start
    pushs "Função que calcula o factorial de 4: "
    writes
    pushi 0     // espaço para o resultado
    pushi 4     // parâmetro
    pusha fact  // fact(4)
    call
    pop 1       // Deixar o resultado na stack
    writei
    writeln.    // writeln(fact(4))
    pushs "That's all folks!"
    writes
stop

fact:
        pushl  -1   // get arg1
        pushi   1
        equal       // n == 1
        jz else
        pushi 1     // Sim
        storel -2   // return(1)
        return
    else:           // Não
        pushl -1    // n * ...
        pushi 0     // fact(n-1)
        pushl -1
        pushi 1
        sub         // n - 1
        pusha fact
        call
        pop 1
        mul         // n * fact(n-1)
        storel -2
        return
```

Desafio: Ler N; Calcular fatorial de N; validar N

Exemplo 9: Resto da divisão inteira pelo algoritmo da diferença

- O programa em Pascal standard seria:

```
program RestoPorSubtracao;

function Resto(a, b: integer): integer;
begin
    while a >= b do
        a := a - b;

    Resto := a;
end;

var
    a, b, r: integer;
begin
    writeln('Introduza a:');
    readln(a);

    writeln('Introduza b:');
    readln(b);

    if b <= 0 then
        writeln('O divisor deve ser positivo.')
    else
        begin
            r := Resto(a, b);
            writeln('O resto da divisão inteira de ', a, ' por ', b, ' é:
', r);
        end;
end.
```

- Vamos centrar-nos na função:

```
function Resto(a, b: integer): integer;
begin
    while a >= b do
        a := a - b;

    Resto := a;
end;
```

- Recebe 2 parâmetros: a e b;
- Produz um resultado.

```
resto:
    ciclo:
        pushl -2
        pushl -1
        supeq         // a < b
        jz fimResto
        pushl -2
        pushl -1
        sub
        storel -2     // a = a - b
        jump ciclo
    fimResto:
        pushl -2
        storel -3     // return a
        return
```

- Juntando tudo:

```
start
    pushi 0
    pushi 0
    pushi 0         // a, b, r: integer;

    pushs "Introduza a: "
    writes
    read
    atoi
    storeg 0

    pushs "Introduza b: "
    writes
    read
    atoi
    storeg 1

    pushg 1
    pushi 0
    inf
    jz bPositivo
    writeln
    pushs "O divisor deve ser positivo."
    writes
bPositivo:
    pushi 0
    pushg 0
    pushg 1
    pusha resto
    call
    pop 2
    storeg 2
    writeln
```

```
    pushs "O resto da divisão inteira de a por b é: "
    writes
    pushg 2
    writei
stop


resto:
    ciclo:
        pushl -2
        pushl -1
        supeq         // a < b
        jz fimResto
        pushl -2
        pushl -1
        sub
        storel -2     // a = a - b
        jump ciclo
    fimResto:
        pushl -2
        storel -3     // return a
        return
```

Exemplo 10: Resto da divisão inteira com uma função recursiva

```
function Resto(a, b: integer): integer;
begin
    if a < b then
        Resto := a
    else
        Resto := Resto(a - b, b);
end;
```

```
resto:
        pushl -2
        pushl -1
        inf         // a < b
        jz fimResto
        pushl -2
        storel -3
        return
    fimResto:
        pushi 0
        pushl -2
        pushl -1
        sub
        pusha resto
        call
```

```
        pop 2
        storel -3     // return a
        return
```
