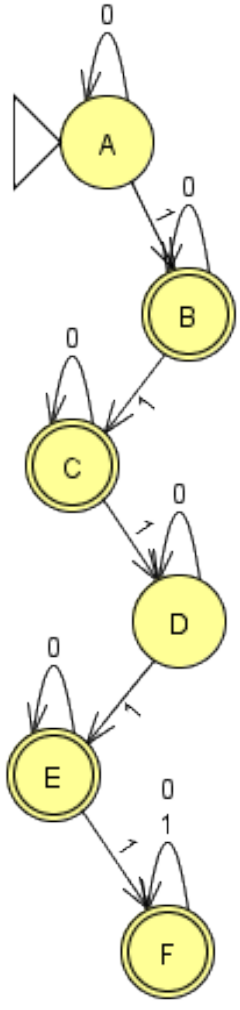


Teoria de Linguagens e Compiladores

Projeto do Compilador

Luiz Eduardo da Silva

Universidade Federal de Alfenas

8 de Março de 2021

Agenda

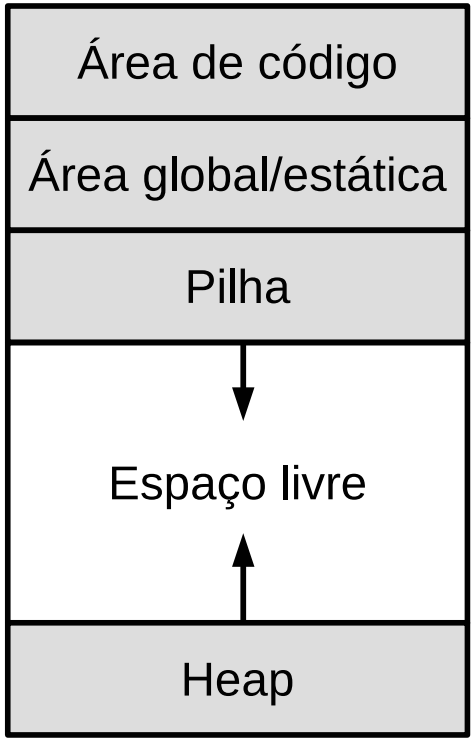
Agenda

Ambientes de Execução

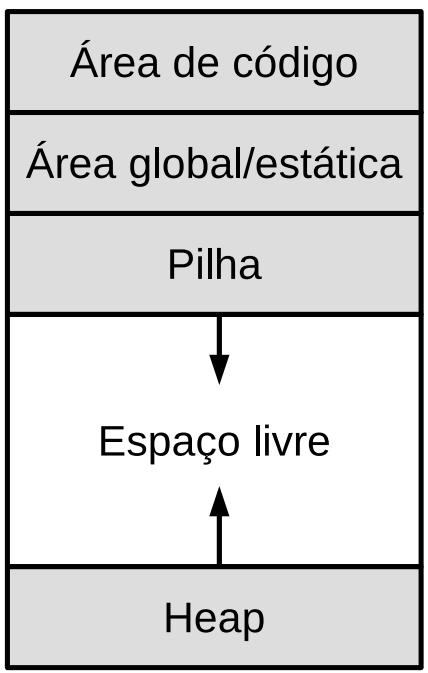
- É necessário um compromisso entre o compilador, o sistema operacional e arquitetura (máquina alvo) para possibilitar que o programa compilado possa ser executado.
- Para isso, o compilador cria uma abstração denominada *ambiente em tempo de execução*, para a qual assume que os programas objeto serão executados.
- Esse ambiente define uma série de questões como:
  - localização, alocação e endereçamento para o programa e os dados do programa
  - mecanismos para acessar variáveis locais e globais.
  - as ligações entre as rotinas.
  - mecanismos de passagem de parâmetro
  - interface com o sistema operacional (E/S, por exemplo)

Organização de Memória

Ambiente de Execução real



Organização de Memória



- Área de código - contém o código do programa objeto e o tamanho é definido durante a compilação.
- Área global/estática - alguns dados como variáveis globais e constantes, também podem ter seus espaços definidos (de forma **estática**) em tempo de compilação.
- Pilha e Heap - são usadas para otimizar a utilização da memória. Nessas áreas são alocados (**dinamicamente**) espaços para as variáveis locais, variáveis dinâmicas, contextos das rotinas que são executadas, etc.

Memória estática e dinâmica

- A localização dos dados na memória em tempo de execução é uma questão importante para a gerência de memória (compilador - SO - arquitetura).
  - **Memória estática** - alocada (reservada) em tempo de compilação
  - **Memória dinâmica** - alocada em tempo de execução.
- Os dados dinâmicos são alocados na Pilha ou Heap.
  - Memória de pilha - dados das rotinas (variáveis locais, parâmetros, endereço de retorno, etc.) são mantidos na pilha de dados.
  - Memória Heap - os dados alocados dinâmicamente (malloc, new, etc.) são mantidos na heap. Para gerenciar a Heap, alguns ambientes de execução implementam rotinas de 'coleta de lixo', que liberam automaticamente espaços alocados na heap e que não são mais acessíveis.

Agenda

Pilha de execução

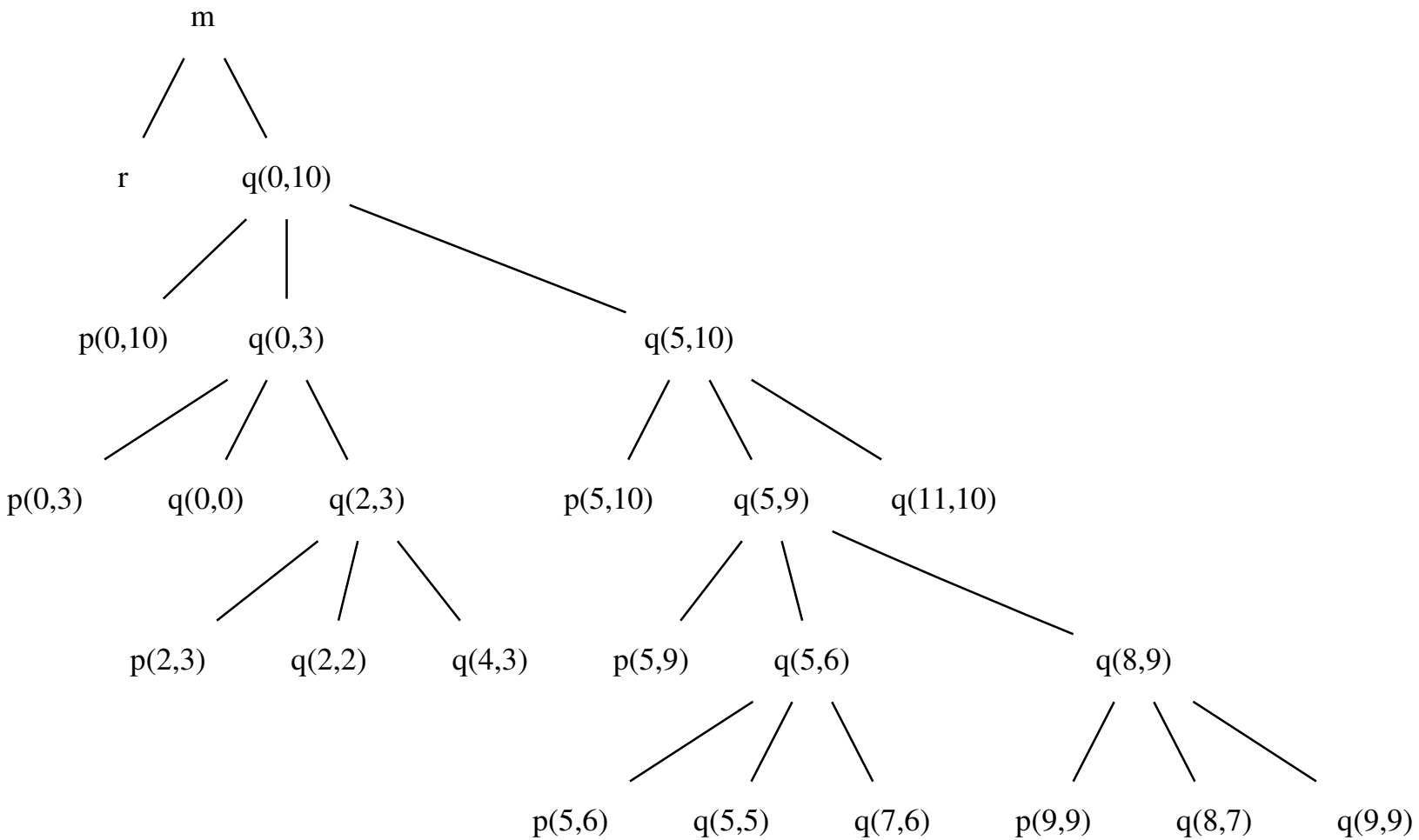
- Os compiladores que implementam **rotinas** (procedimento, função, métodos, etc...), utilizam a Memória de Pilha para gerenciar a execução dessas unidades de código.
- Quando a rotina é chamada, um espaço da Memória de Pilha é alocado para seus parâmetros, variáveis locais, contexto para retornar a execução no ponto em que a rotina foi chamada, etc.
- Esse espaço é desalocado logo que a rotina termina.
- Os endereços relativos dos dados da rotina são os mesmos, independente da sequência de chamadas.
- O uso da pilha só é viável porque as chamadas das rotinas se aninham durante a execução do programa.

Árvore de ativação (exemplo)

```
int a[11];
void readArray() {
  int i;
  ...
}
int partition(int m, int n) {
  ...
}
int quicksort(int m, int n) {
  int i;
  if (n > m) {
    i = partition (m, n);
    quicksort (m, i-1);
    quicksort (i+1, n);
  }
}
main() {
  readArray();
  quicksort (0,10);
}
```

```
entra em main()
  entra em readArray()
  sai do readArray()
  entra em quicksort (0,10)
    entra em partition (0,10)
    sai do partition (0,10)
    entra em quicksort (0,3)
        ...
    sai do quicksort (0,3)
    entra em quicksort (5,10)
        ...
    sai do quicksort (5,10)
  sai do quicksort (0,10)
sai do main()
```

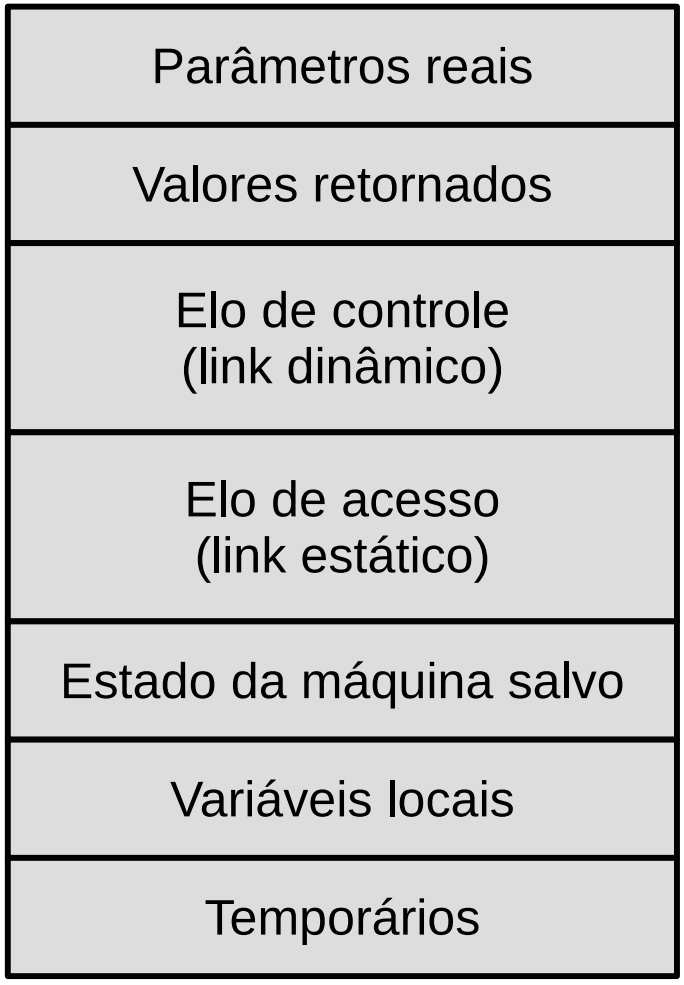
Árvore de ativação (exemplo)



Árvore de ativação

- Podemos representar a sequência de chamadas das rotinas na execução de um programa através de uma estrutura de **árvore de ativação**.
  - Cada nó é uma ativação (chamada de uma rotina). Os filhos de uma raiz representam as rotinas que são ativadas na execução da rotina pai.
  - Um filho tem que finalizar para que o irmão da direita possa começar a execução.
- A pilha de execução é possível pois:
  - A sequência de chamadas corresponde a um percurso em pré-ordem na árvore de ativação.
  - A sequência de retornos corresponde a um percurso em pós-ordem na árvore.
  - Para uma ativação de uma rotina filha F. As rotinas ativas (vivas) são aquelas no caminho da raiz R da árvore até a rotina F (em ordem inversa na pilha, de F para R).

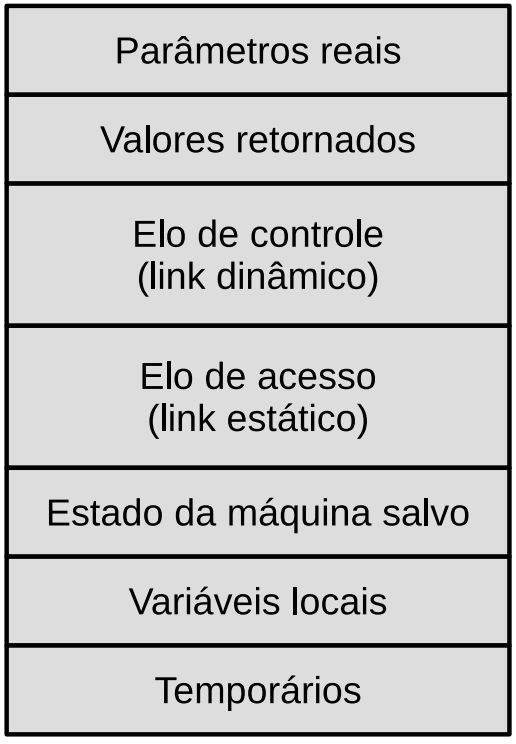
Registro de ativação



Registro de ativação

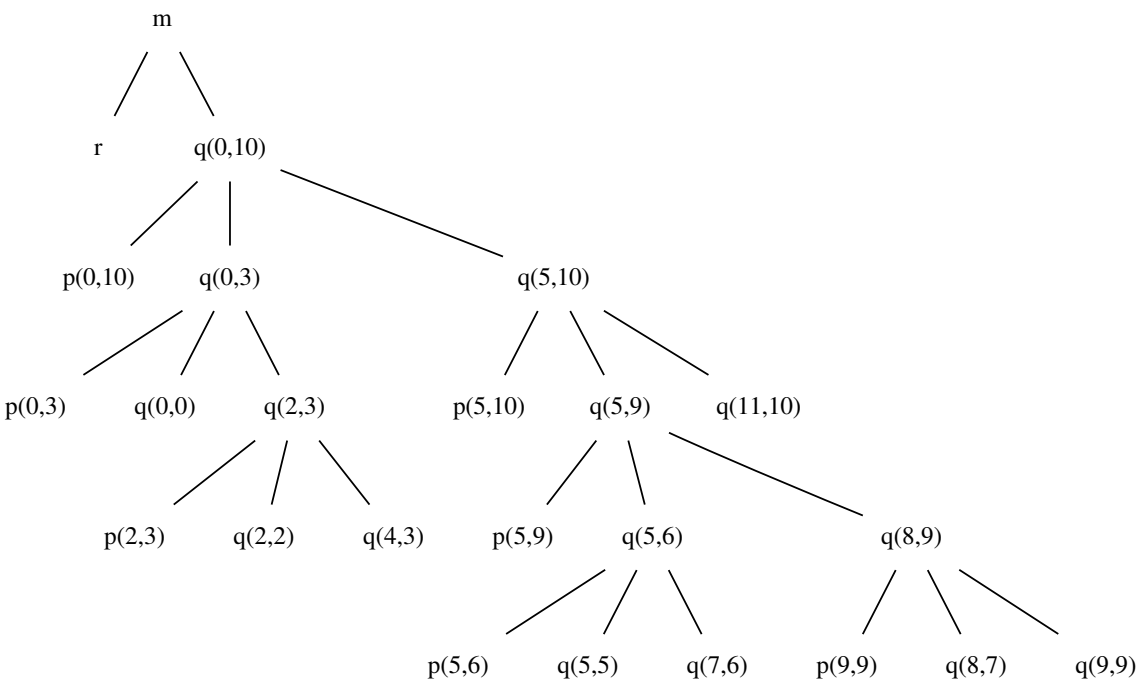
- As chamadas e os retornos das rotinas são gerenciados na pilha de execução.
- Cada rotina em execução (ativação viva) mantém algumas informações na pilha.
- Ao conjunto de informações da rotina em execução na pilha damos o nome de **registro de ativação** (ou frame).
- A pilha contém os registros de ativação das rotinas vivas, correspondendo ao caminho na árvore de ativação da Raiz até a rotina em execução.
- O registro de ativação da rotina em execução aparece no topo da pilha.

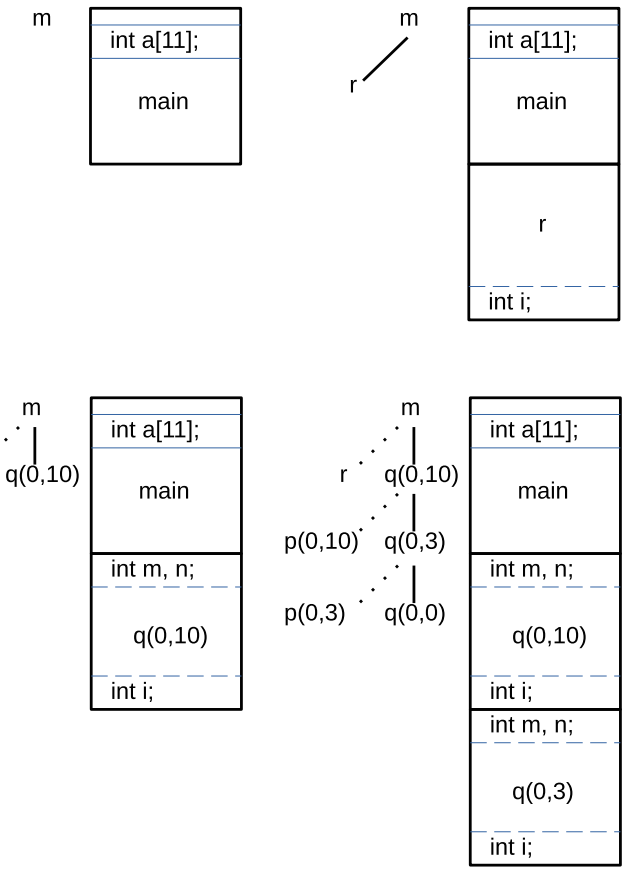
Registro de ativação



- Temporários - valores usados na avaliação de expressões.
- Variáveis locais - variáveis da rotina do registro.
- Estado da máquina salvo - contexto da máquina quando a rotina foi chamada (endereço de retorno para o chamador, por exemplo)
- Elo de acesso - usado para localizar dados em outro registro de ativação.
- Elo de controle - aponta o registro de ativação da rotina chamadora.
- Valores retornados - local onde devem ser armazenados os valores retornados da rotina (após a execução)
- Parâmetros reais - parâmetros da rotina chamada (normalmente usa registradores), mas de forma geral pode usar a pilha de execução.

Registros na pilha de execução



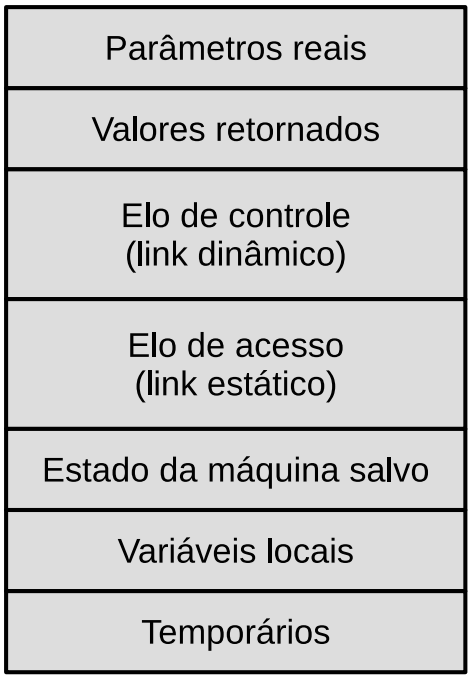


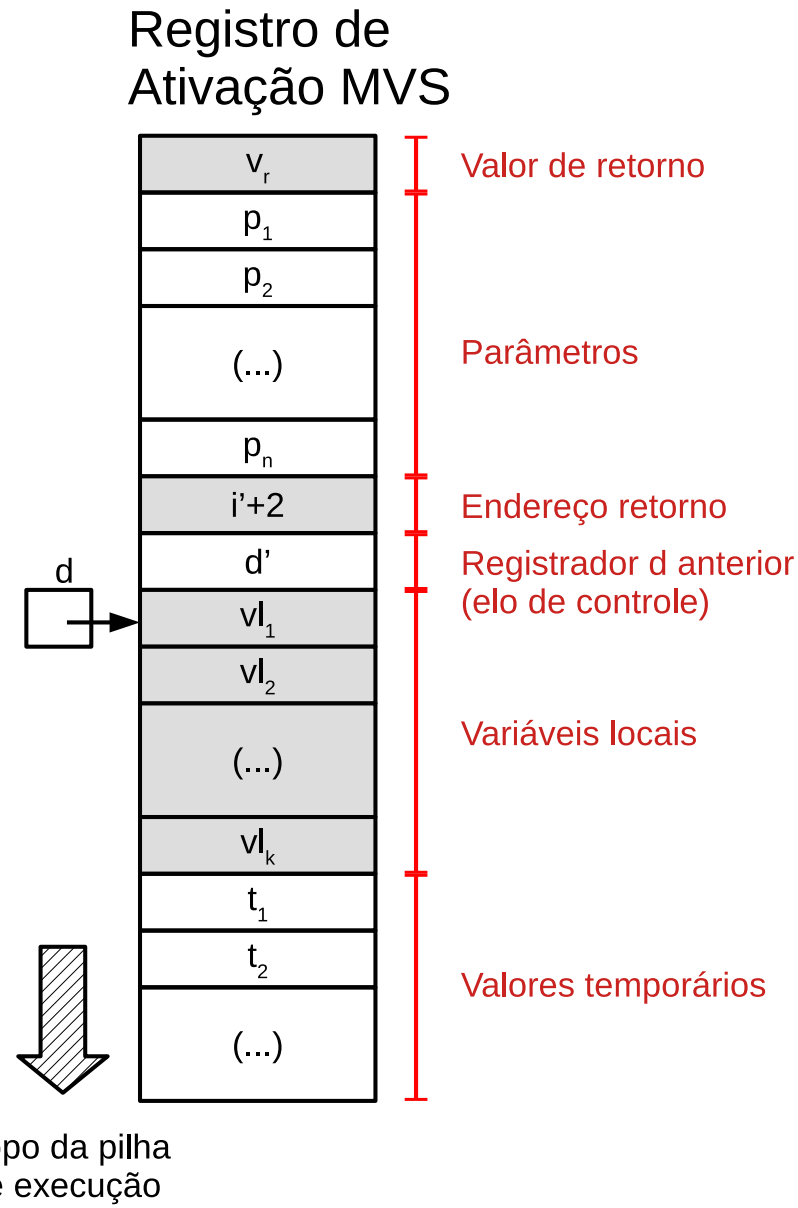
Agenda

Instruções necessárias

Instrução	Descrição	Micro-código
CRVL n	Carrega valor local	$s \leftarrow s + 1$ $M[s] \leftarrow M[d + n]$
ARZL n	Armazena Local	$M[d + n] \leftarrow M[s]$ $s \leftarrow s - 1$
CREL n	Carrega endereço local	$s \leftarrow s + 1$ $M[s] \leftarrow d + n$
CREG n	Carrega endereço global	$s \leftarrow s + 1$ $M[s] \leftarrow n$
CRVI n	Carrega valor indireto	$s \leftarrow s + 1$ $M[s] \leftarrow M[M[d + n]]$
ARMI n	Armazena Indireto	$M[M[d + n]] \leftarrow M[s]$ $s \leftarrow s - 1$
SVCP	Salva Contador do Programa	$s \leftarrow s + 1$ $M[s] \leftarrow i + 2$
ENSP	Entrada sub-programa	$s \leftarrow s + 1$ $M[s] \leftarrow d$ $d \leftarrow s + 1$
RTSP n	Retorno sub-programa	$d \leftarrow M[s]$ $i \leftarrow M[s - 1]$ $s \leftarrow s - (n + 2)$
DMEM n	Desaloca memória	$s \leftarrow s - n$

Registro de ativação MVS





Regras para rotinas

#	Regra
1	programa: cabecalho variaveis **rotinas** T_INICIO lista_comandos T_FIM
	(...)
12	**rotinas:** λ
13	| **lista_rotinas**
14	**lista_rotinas: lista_rotinas rotina**
15	| **rotina**
16	**rotina: funcao**
17	| **procedimento**
18	**funcao: T_FUNC tipo identificador T_ABRE lista_parametros T_FECHA variaveis T_INICIO lista_comandos T_FIMFUNC**
19	**procedimento: T_PROC identificador T_ABRE lista_parametros T_FECHA variaveis T_INICIO lista_comandos T_FIMPROC**
20	**lista_parametros:** λ
21	| **parametro lista_parametros**
22	**parametro: mecanismo tipo identificador**
23	**mecanismo:** λ| **T_REF**
	(...)

Regras para rotinas

#	Regra
	(...)
27	comando :
	(...)
31	| **chamada_procedimento**
	(...)
39	**chamada_procedimento: identificador T_ABRE lista_argumentos T_FECHA**
40	**lista_argumentos: lista_argumentos argumento**
41	| λ
42	**argumento: expressao**
	(...)
53	**chamada:** λ
54	| **T_ABRE lista_argumentos T_FECHA**
55	termo: identificador **chamada**
	(...)

Modificação da Tabela de Símbolos

Tabela de Símbolos

#	**id**	**esc**	**dsl**	**rot**	cat	tip	mec	par

- # - identifica a posição do símbolo na tabela
- **nome(id)** - é o nome do identificador, o nome escolhido pelo programador para a entidade do programa
- **escopo(esc)** - escopo da variável. No caso da linguagem simples os símbolos só podem ter escopo global (G) ou local (L).
- **deslocamento(dsl)** - deslocamento (ou endereço) é o operando que acompanhará as variáveis na instrução CRVG, CRVL. As funções também tem valor para esse campo.
- **rótulo(rot)** - rótulo atribuído pelo compilador para a função ou procedimento. Essa informação é necessária para tradução da instrução de desvio (DSVS) na chamada do procedimento ou função.

Modificação da Tabela de Símbolos

Tabela de Símbolos

#	id	esc	dsl	rot	**cat**	**tip**	**mec**	**par**

- **categoria(cat)** - categoria do símbolo que pode ser: VAR = variável, PRO = procedimento, FUN = função, PAR = parâmetro.
- **tipo(tip)** - tipo do símbolo (INT = inteiro ou LOG = lógico).
- **mecanismo(mec)** - mecanismo de passagem para parâmetros que pode ser REF = referência ou VAL = valor.
- **parâmetros(par)** - lista encadeada dos parâmetros da rotina, com seus Tipos e Mecanismos de passagem de parâmetro. Essa informação é necessária para traduzir de forma correta os parâmetros na chamada da rotina. Observe que no programa principal as variáveis locais e parâmetros das rotinas são excluídos da tabela de símbolos

Tabela de Simbolos (exemplo)

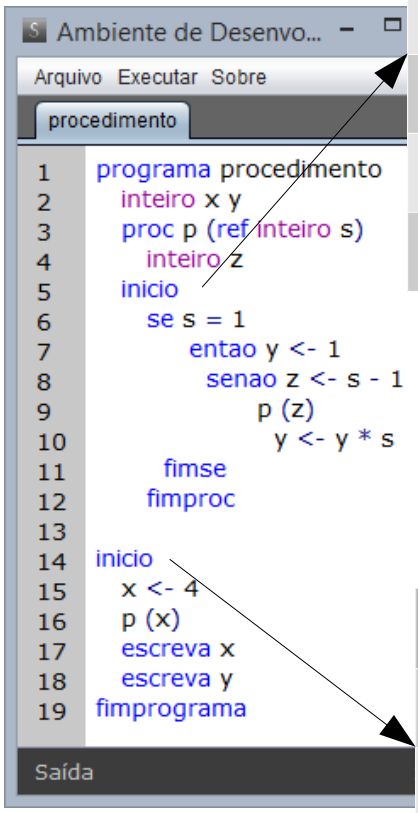


Tabela de Símbolos

#	Id	Esc	Dsl	Rot	Cat	Tip	Mec	Par
0	x	G	0		VAR	INT		
1	y	G	1		VAR	INT		
2	p	G	-	1	PRO	-		
3	s	L	-3		PAR	INT	REF	
4	z	L	0		VAR	INT		

Tip	Mec	Prox
INT	REF	

Tabela de Símbolos

#	Id	Esc	Dsl	Rot	Cat	Tip	Mec	Par
0	x	G	0		VAR	INT		
1	y	G	1		VAR	INT		
2	p	G	-	1	PRO	-		

Tip	Mec	Prox
INT	REF	

Exemplo de tradução

```
programa parametros
  proc somatudo (inteiro a
                 inteiro b
                 inteiro c
                 inteiro d)
  inicio
    escreva a + b + c + d
  fimproc

inicio
  somatudo (10 3 4 8)
fimprograma
```

```
    INPP
    DSVS  L0
L1  ENSP
    CRVL  -6
    CRVL  -5
    SOMA
    CRVL  -4
    SOMA
    CRVL  -3
    SOMA
    ESCR
    RTSP  4
L0  NADA
    CRCT  10
    CRCT  3
    CRCT  4
    CRCT  8
    SVCP
    DSVS  L1
    FIMP
```

Simulação da Pilha de Execução

```
1  programa parametros
2    proc somatudo (inteiro a
3                   inteiro b
4                   inteiro c
5                   inteiro d)
6    inicio
7      escreva a + b + c + d
8    fimproc
9
10 inicio
11   somatudo (10 3 4 8)
12 fimprograma
```

Vetor P

#	Rótulo	Código	Operandos
0		INPP	
1		DSVS	L0
2	L1	ENSP	
3		CRVL	-6
4		CRVL	-5
5		SOMA	
6		CRVL	-4
7		SOMA	
8		CRVL	-3
9		SOMA	
10		ESCR	
11		RTSP	4
12	L0	NADA	
13		CRCT	10
14		CRCT	3
15		CRCT	4
16		CRCT	8
17		SVCP	
18		DSVS	L1
19		FIMP	

Registradores

i: 3 s: 5 d: 6

Pilha M

#	Valor	
0	10	
1	3	
2	4	Parâmetros
3	8	
4	19	Retorno (i+2)
5	-1	Base (d')

Exemplo de tradução

```
programa recursao
  inteiro x

proc recursivo (inteiro n)
inicio
  se n > 0
    entao escreva n
          recursivo (n - 1)
    senao
  fimse
fimproc

inicio
  leia x
  recursivo (x)
fimprograma
```

```
      INPP
      AMEM  1
      DSVS  L0
L1    ENSP
      CRVL  -3
      CRCT  0
      CMMA
      DSVF  L2
      CRVL  -3
      ESCR
      CRVL  -3
      CRCT  1
      SUBT
      SVCP
      DSVS  L1
      DSVS  L3
L2    NADA
L3    NADA
      RTSP  1
L0    NADA
      LEIA
      ARZG  0
      CRVG  0
      SVCP
      DSVS  L1
      DMEM  1
      FIMP
```

Simulação da Pilha de Execução

```
1  programa recursao
2    inteiro x
3
4  proc recursivo (inteiro n)
5  inicio
6    se n > 0
7      entao escreva n
8            recursivo (n - 1)
9      senao
10   fimse
11 fimproc
12
13 inicio
14   leia x
15   recursivo (x)
16 fimprograma
```

Vetor P

#	Rótulo	Código	Operandos
2		DSVS	L0
3	L1	ENSP	
4		CRVL	-3
5		CRCT	0
6		CMMA	
7		DSVF	L2
8		CRVL	-3
9		ESCR	
10		CRVL	-3
11		CRCT	1
12		SUBT	
13		SVCP	
14		DSVS	L1
15		DSVS	L3
16	L2	NADA	
17	L3	NADA	
18		RTSP	1
19	L0	NADA	
20		LEIA	
21		ARZG	0
22		CRVG	0
23		SVCP	
24		DSVS	L1
25		DMEM	1

Registradores

i: 3 s: 11 d: 10

Pilha M

#	Valor	
0	3	
1	3	Recursivo(3)
2	25	
3	-1	
4	2	Recursivo(2)
5	15	
6	4	
7	1	Recursivo(1)
8	15	
9	7	
10	0	Recursivo(0)
11	15	

Exemplo de tradução

```
programa referencia
inteiro x

proc muda(ref inteiro a)
inicio
  a <- 7
fimproc

inicio
  muda (x)
  escreva x
fimprograma
```

```
     INPP
     AMEM 1
     DSVS L0
L1   ENSP
     CRCT 7
     ARMI -3
     RTSP 1
L0   NADA
     CREG 0
     SVCP
     DSVS L1
     CRVG 0
     ESCR
     DMEM 1
     FIMP
```

Exemplo de tradução

```
programa testafatorial
  inteiro n

  func inteiro fatorial (inteiro n)
  inicio
    se n = 0
       entao fatorial <- 1
       senao fatorial <- n * fatorial (n - 1)
    fimse
  fimfunc

inicio
  leia n
  escreva fatorial (n)
fimprograma
```

```
      INPP
      AMEM 1
      DSVS L0
L1    ENSP
      CRVL -3
      CRCT 0
      CMIG
      DSVF L2
      CRCT 1
      ARZL -4
      DSVS L3
L2    NADA
      CRVL -3
      AMEM 1
      CRVL -3
      CRCT 1
      SUBT
      SVCP
      DSVS L1
      MULT
      ARZL -4
L3    NADA
      RTSP 1
L0    NADA
      LEIA
      ARZG 0
      AMEM 1
      CRVG 0
      SVCP
      DSVS L1
      ESCR
      DMEM 1
      FIMP
```