unesp

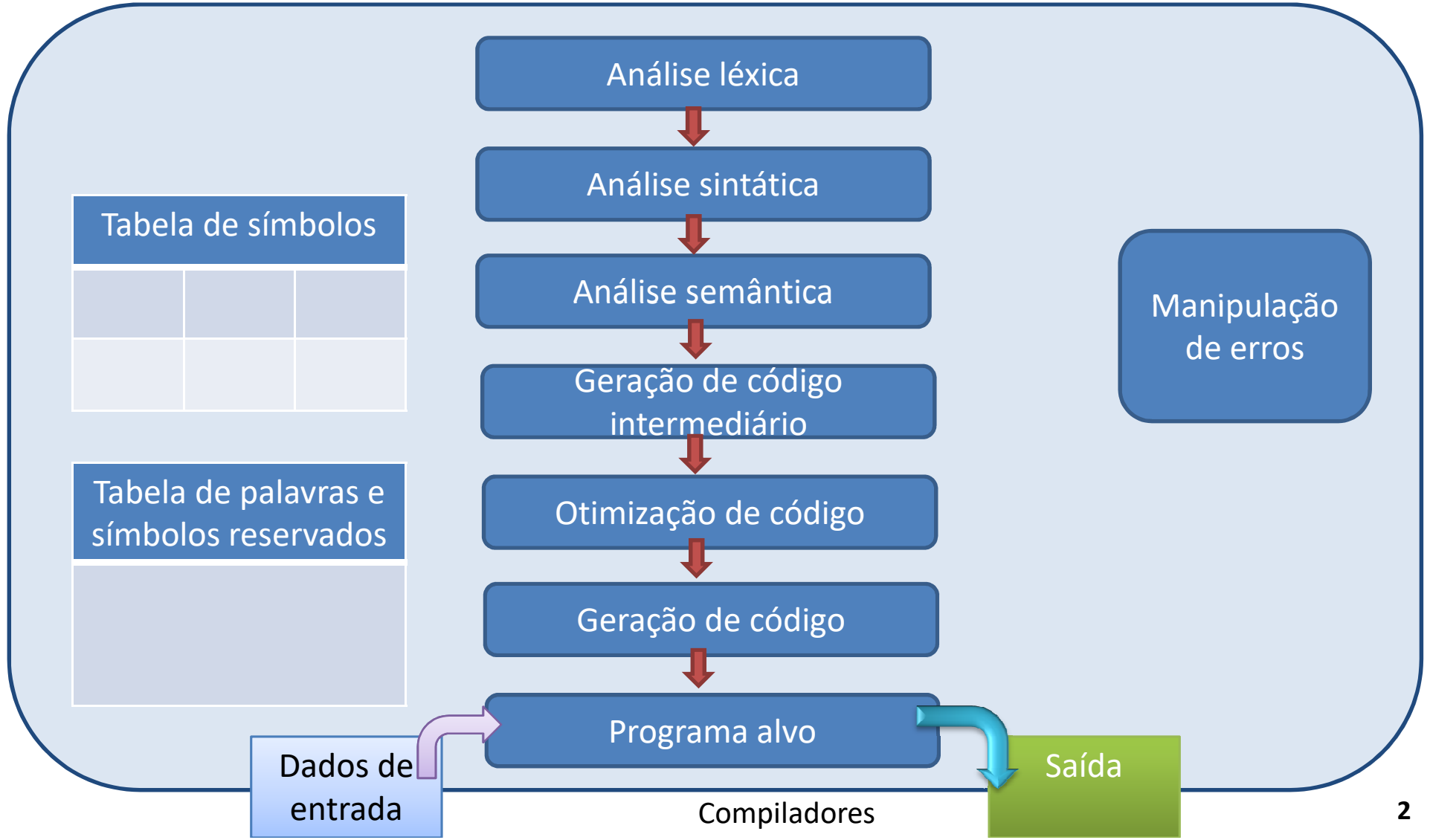
Compiladores

Aula 13

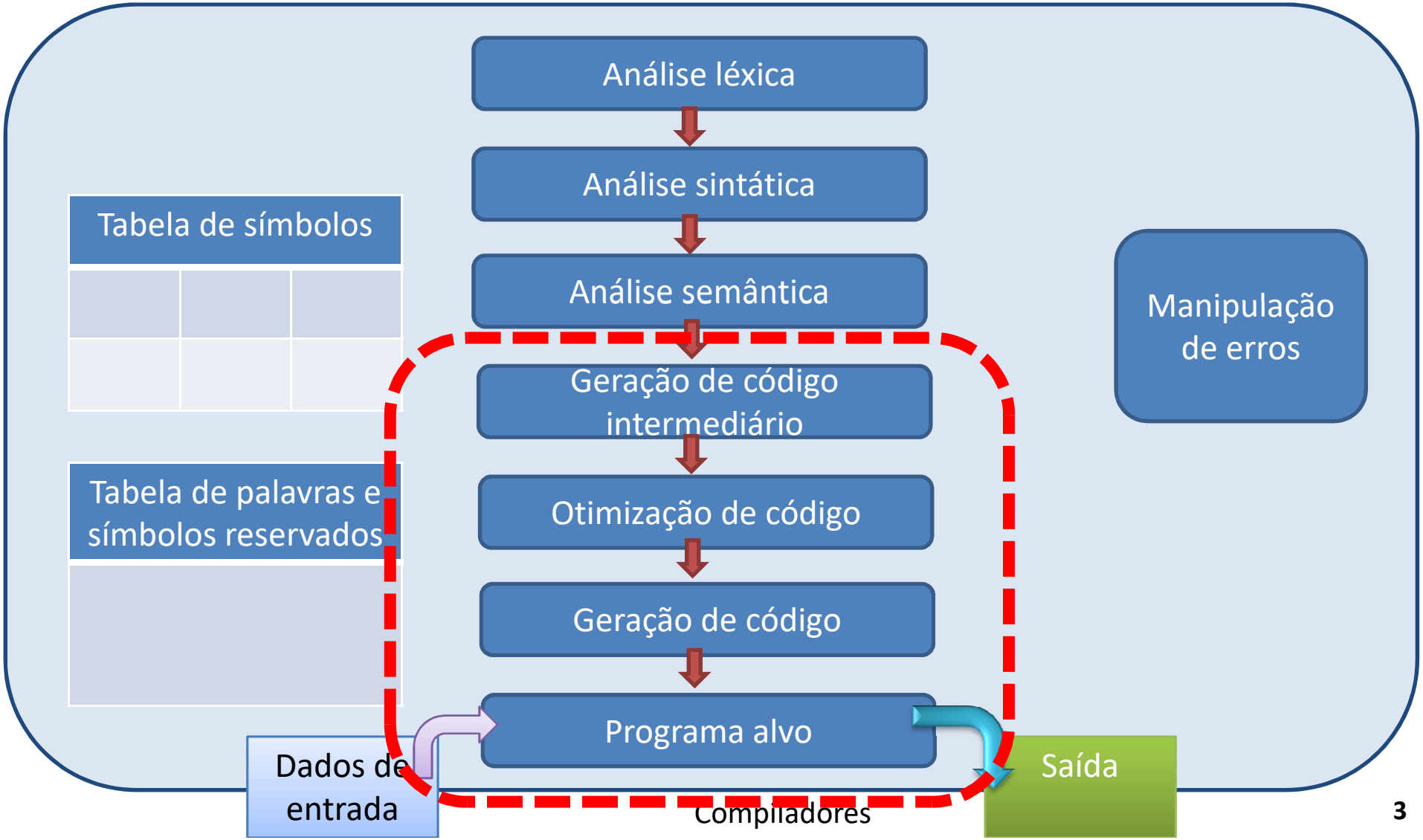
Celso Olivete Júnior

olivete@fct.unesp.br

Na aula de hoje



Na aula de hoje



```java
public class JavaApplication8 {

    /**
     * @param args the command line arguments
     */
    /*-comentario do codigo fonte*/
    public static void main(String[] args) {
        // TODO code application logic here

    int i =0;
    double j = 0;
    while (i<10)
     {
         i = i + 1;
         j = j + 1;
     }

    }
}
```

Instrucoes2 - Bloco de notas

Arquivo Editar Formatar Exibir Ajuda

```
Compiled from "JavaApplication8.java"
public class javaapplication8.JavaApplication8 {
  public javaapplication8.JavaApplication8();
    Code:
       0: aload_0
       1: invokespecial #1 // Method java/lang/Object."<init>":()
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: iconst_0
       1: istore_1
       2: dconst_0
       3: dstore_2
       4: iload_1
       5: bipush        10
       7: if_icmpge     21
      10: iload_1
      11: iconst_1
      12: iadd
      13: istore_1
      14: dload_2
      15: dconst_1
      16: dadd
      17: dstore_2
      18: goto          4
      21: return
}
```

Geração de código intermediário

- Objetivos
  - Gerar o *Bytecode* (código de montagem) para LALG
    - Formato de código intermediário entre o código fonte, o texto que o programador consegue manipular, e o código de máquina, que o computador consegue executar.
  - Requer interpretação posterior
    - Executar o *Bytecode* com a máquina virtual MEPA

Geração de código para LALG

- Geração de código será feita de forma *Ad hoc,* atrelada aos procedimentos sintáticos
  - Geração diretamente dos procedimentos sintáticos ou
  - Via chamada a procedimentos/funções de geração com argumentos específicos
    - Tabela de símbolos dará o suporte necessário para a geração

Geração de código intermediário

MEPA: MÁQUINA DE EXECUÇÃO PARA PASCAL (OU C)

- É uma máquina virtual para execução dos programas escritos em Pascal Simplificado. Dado um programa, o objetivo será apresentar as instruções *assembly* que causam o efeito esperado

MEPA - Arquitetura

- Máquina a pilha (Stack Machine), dividida em:

  Área de Código– simulado em um vetor (C)

  Área de Dados - simulado em um vetor (D)

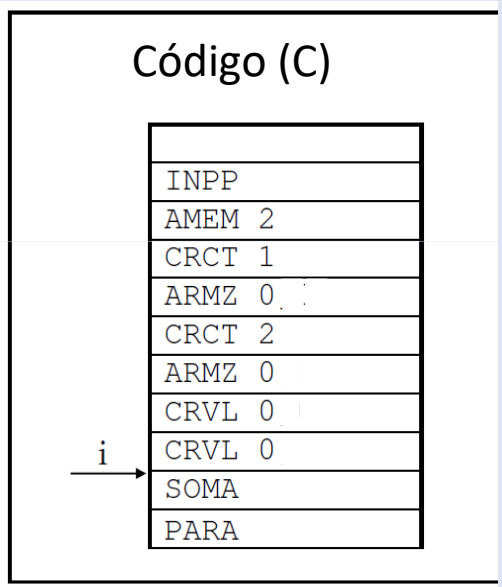
- Dois registradores

  Contador de programa - i

  Topo da Pilha - s

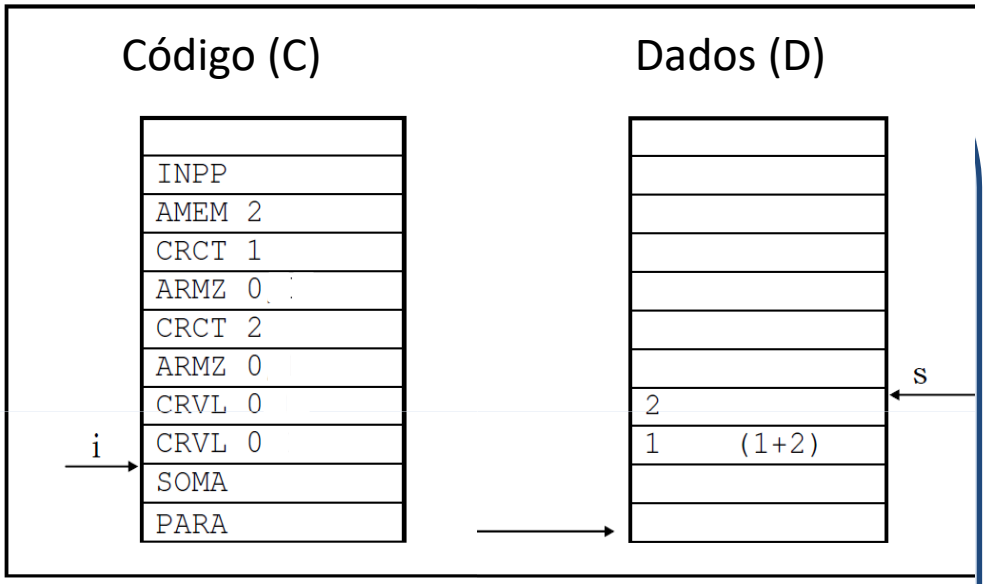
Geração de código para LALG

- Área de Código (C)
- A geração consiste no preenchimento deste vetor conforme o programa for sendo compilado
  - Posição atual marcada por C[i]
  - Ao final, o vetor C é gravado em um arquivo de saída
    - As instruções não podem ir direto para o arquivo conforme forem sendo geradas
      - Ainda poderão ser modificadas
      - O processo pode parar por algum motivo e o tempo de escrita em arquivo seria perdido



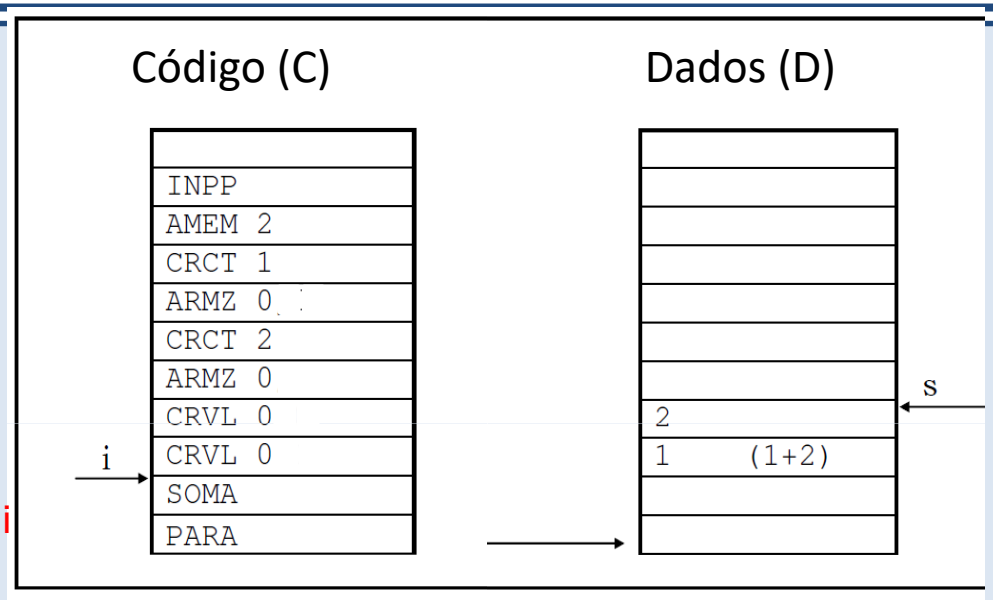
Geração de código para LALG

- Área de Dados (D)



  - Vetor que se comporta como uma pilha
  - Topo marcado por D[s]
  - Só existirá realmente durante a execução
  - As instruções funcionarão sobre seu topo e, muitas vezes, sobre a posição imediatamente abaixo do topo (no caso de uma operação binária)

MEPA – Arquitetura e interpretação



Contador de programa - i

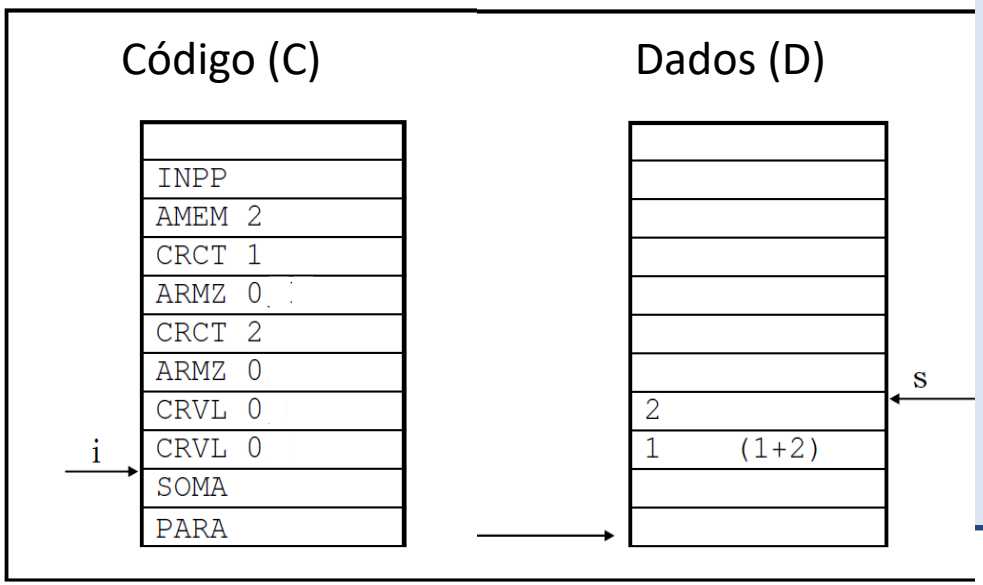
Topo da Pilha - s

Micro código da CPU

```
Enquanto (i válido and (not FIM) ) do
        Executa C[i]
        Se C[i] indica desvio
                então i := endereço do desvio
                senão i := i + 1
        FimSe
FimEnquanto
```

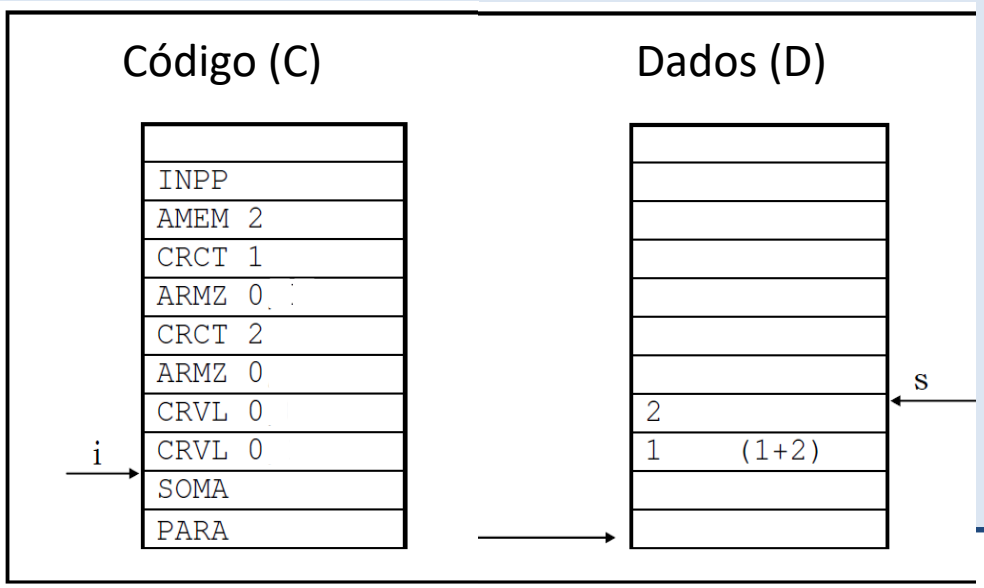
Geração de código
Vetor de código (C)

- O código intermediário é gerado em conjunto com a análise sintática
  - Registrador ***i*** indica a posição atual;
  - Cada posição contém uma instrução - gerada de acordo as instruções da MEPA;
  - No final da compilação C[i] conterá o código intermediário.



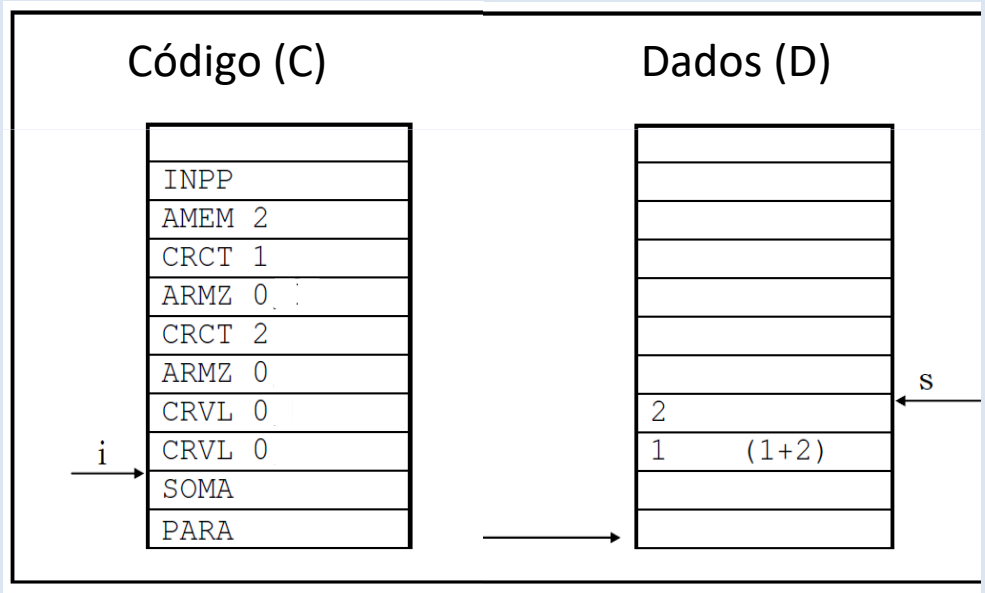
Geração de código
Vetor de código (C)

- O código intermediário é gerado em conjunto com a análise sintática
  - Necessário acrescentar na tabela de símbolos um novo atributo
    - Endereço relativo de um identificador na pilha (D)



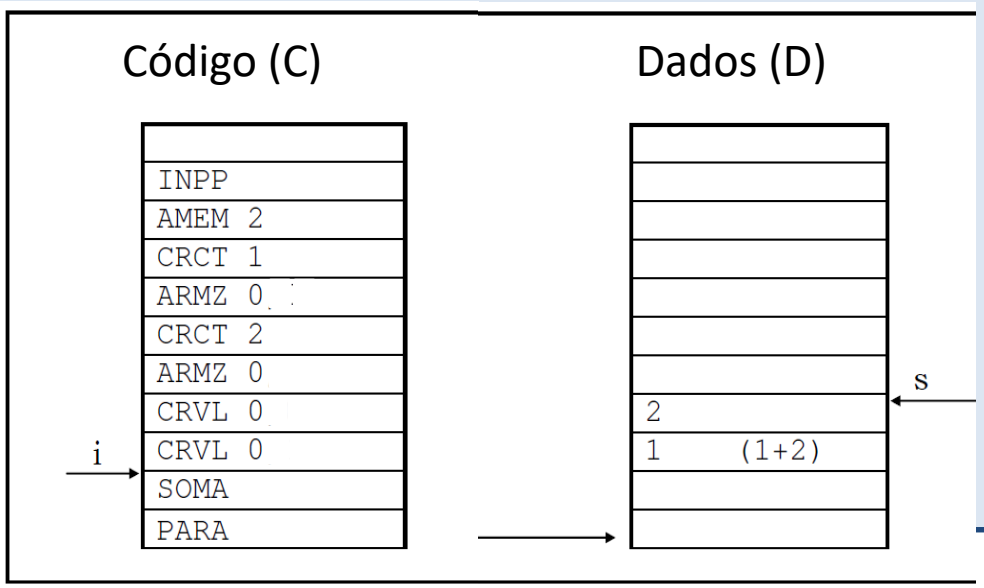
Geração de código
Vetor de código (C)

- O código intermediário armazenado no vetor C será interpretado posteriormente



Geração de código
Vetor de dados (D)

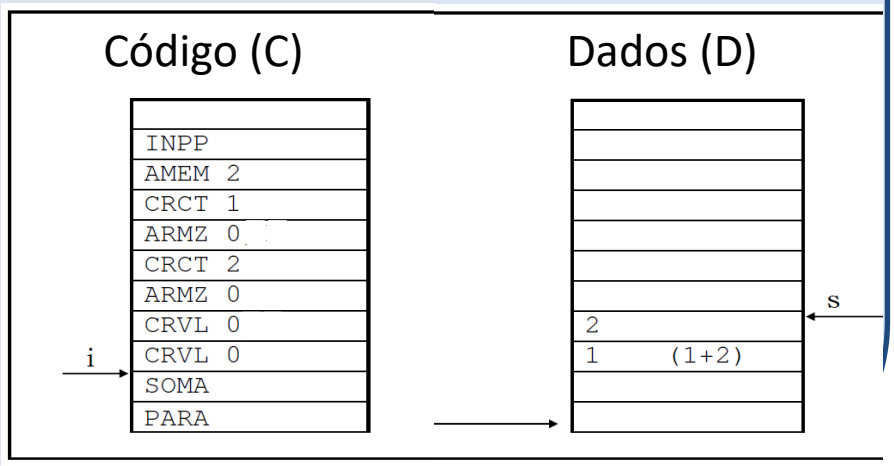
- Características
  - Vetor que se comporta como uma pilha
    - Topo acessado via D[s]
  - Só existe em tempo de execução
  - Instruções operam sobre o topo (e/ou topo -1)



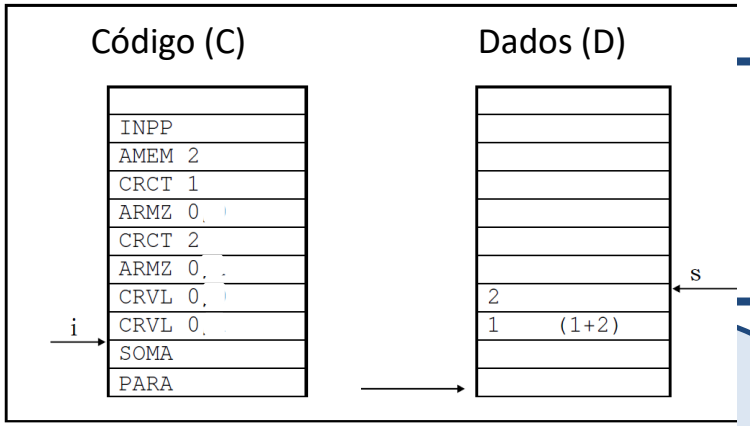
Interpretação do código

- O código armazenado no vetor C é executado a partir de um interpretador
  - Instruções C[i] são executadas, manipulando os dados do vetor D, até encontrar uma instrução de parada (neste caso, a inst. **PARA**) ou algum erro
    - Conforme as instruções são executadas, a pilha é manipulada

- A execução de cada instrução aumenta de 1 o valor de i, exceto as instruções que envolvem desvios
- Como só há os tipos inteiro e real na LALG, a pilha D pode ser um vetor de reais



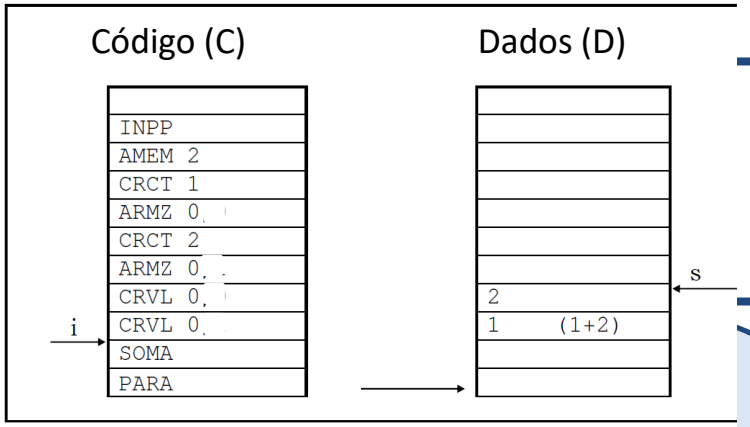
MEPA
Instruções



Significado	Instrução	Execução
Carrega Constante **c** no Topo (pilha)	**CRCT c**	s:=s+1; D[s]:=c;
Carrega valor de end. **n** no Topo	**CRVL n**	s:=s+1; D[s] := D[n];
Armazena valor do Topo no end. **n** de D	**ARMZ n**	D[n] := D[s]; s := s - 1;
Soma	**SOMA**	D[s-1]:=D[s-1]+D[s]; s:=s-1;
Subtração (**SUBT**), Multiplicação (**MULT**), Divisão (**DIVI**) e Resto da divisão Inteira (**MODI**) – Idem a **SOMA**		

Efetua a soma dos dois valores do topo da pilha. Desempilha os dois valores do topo da pilha e empilha o resultado no topo da pilha

MEPA
Instruções



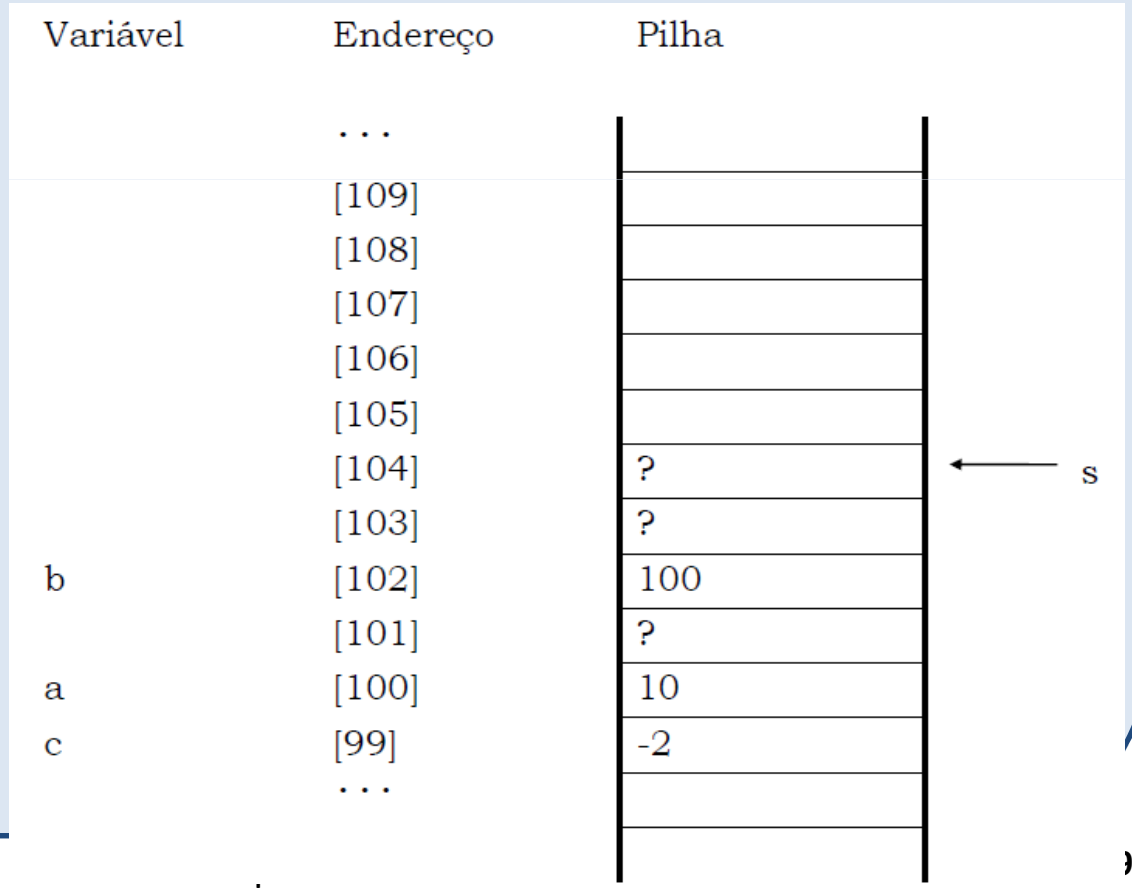
Significado	Instrução	Execução
Inverte Sinal do Topo	**INVR**	D[s]:= – D[s];
Conjunção (**and**) de valores lógicos -> Falso=0; Verdadeiro=1	**CONJ**	**Se** D[s-1]=1 **and** D[s]=1 **então** D[s-1]:=1; **senão** D[s-1]:=0; s:=s-1;
Disjunção (**or**) de valores lógicos	**DISJ**	**Se** D[s-1]=1 **or** D[s]=1 **então** D[s-1]:=1; **senão** D[s-1]:=0; s:=s-1;
Negação (**not**)	**NEGA**	D[s]:= 1 - D[s]
Compara se Menor	**CMME**	**Se** D[s-1] < D[s] **então** D[s-1]:=1 **senão** D[s-1]:=0; s:=s-1;

Compara se Maior (**CMMA**), Compara se Igual (**CMIG**), Compara se Desigual (**CMDG**), Compara se Maior ou Igual (**CMAG**) e Compara se Menor ou Igual (**CMEG**) – todas instr. idem a **CMME**

MEPA – exemplo 1

- a = a + b * c
- Considerando a tabela de símbolos com os atributos endereço de memória e valor.

ID	Endereço	Valor
a	[100]	10
b	[102]	100
c	[99]	-2
s		104



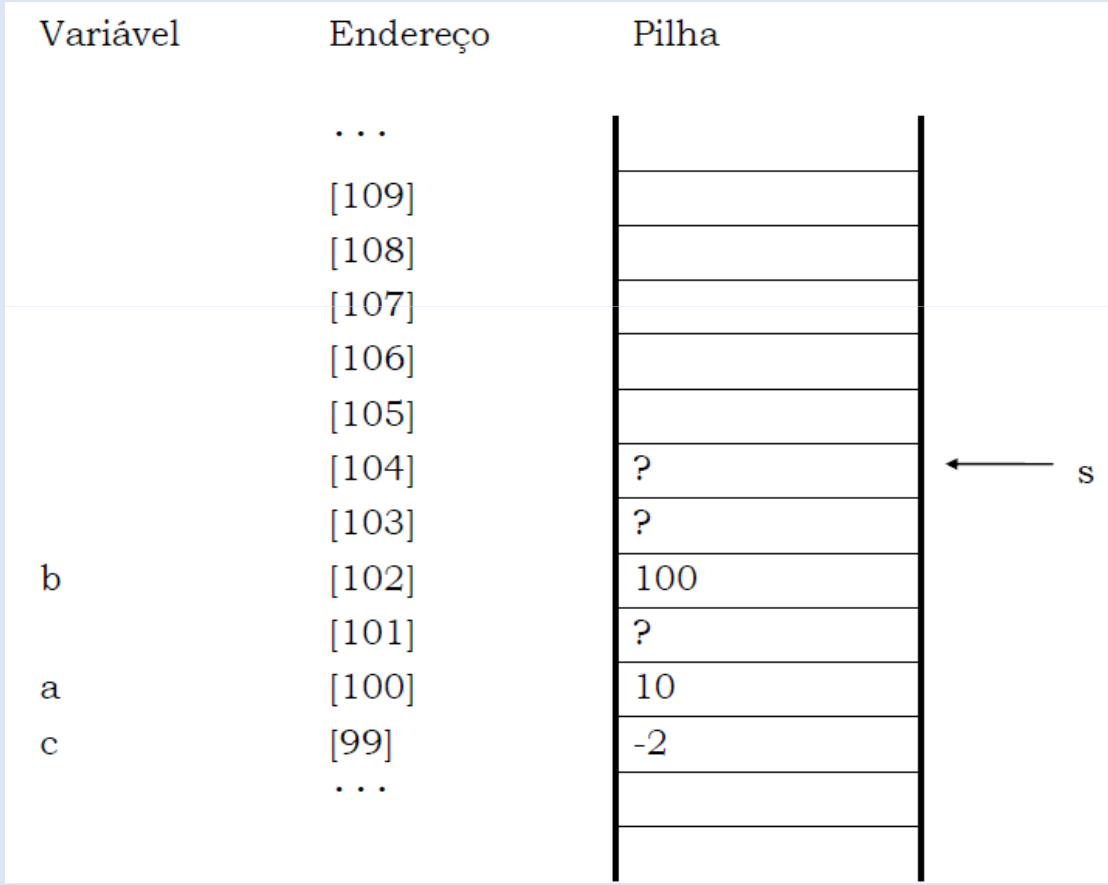
MEPA
Exemplo de Comando de atribuição

- a = a + b * c

- Tradução
  - **CRVL a***
  - **CRVL b***
  - **CRVL c***
  - **MULT**
  - **SOMA**
  - **ARMZ a***

*** endereço na Pilha obtido na Tabela de Símbolos**

ID	Endereço	Valor
a	[100]	10
b	[102]	100
c	[99]	-2
s		104



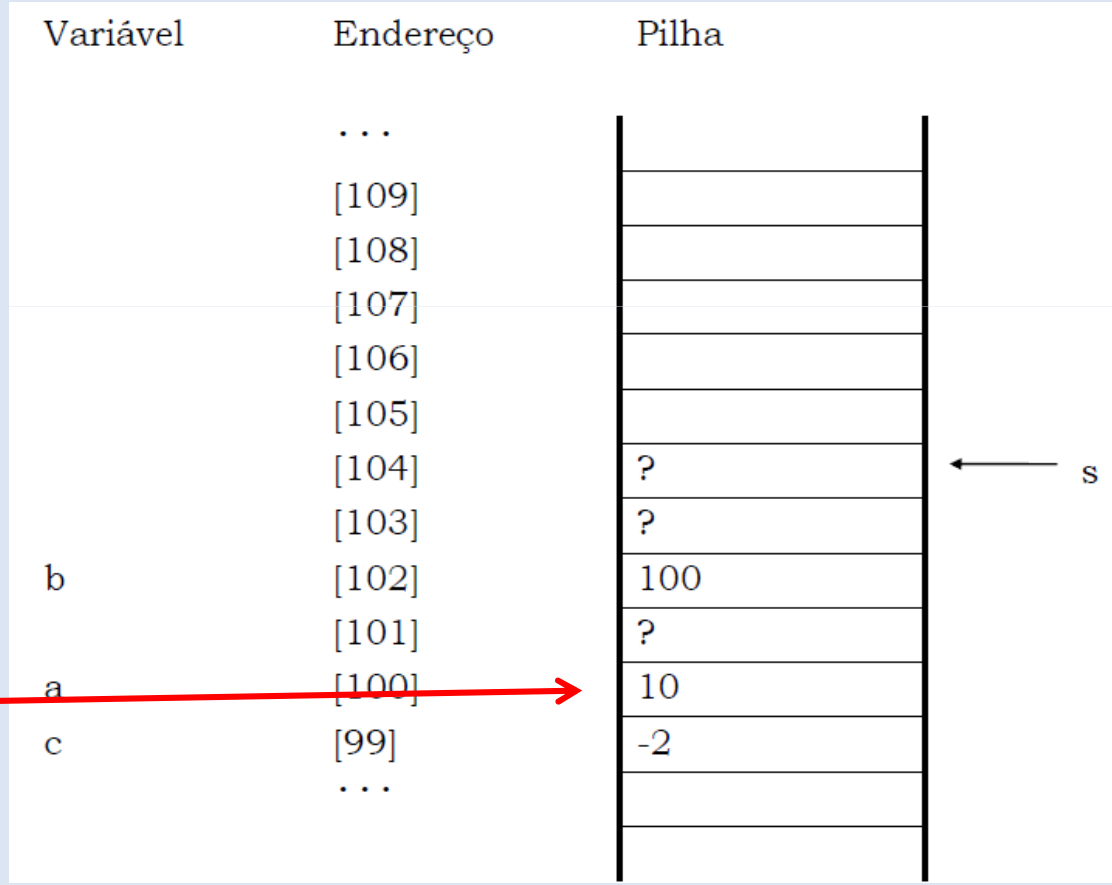
MEPA
Comando de atribuição

- a = a + b * c

- Tradução

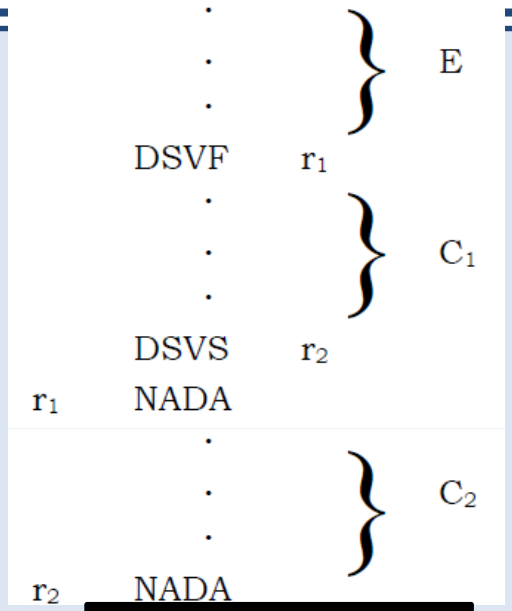
CRVL a*
CRVL b*
CRVL c*
MULT
SOMA
ARMZ a*

ID	Endereço	Valor
a	[100]	10
b	[102]	100
c	[99]	-2
s		104



MEPA
Comandos condicionais

- if E **then** C1 **else** C2
- Tradução



Novas instruções

Significado	Instrução	Execução
Desvia sempre para a instr. de endereço p	**DSVS p**	i:=p;
Desvia se Falso	**DSVF p**	se D[s]=0 **então i:=p** **senão i:=i+1;** s:=s-1;
Comando Nulo	**NADA**	{}

MEPA
Comandos condicionais

- Exemplo: **if (q)**

then a:= 1

else a := 2

Tradução

```
      CRVL q*
      DSVF L1
      CRCT 1
      ARMZ a*
      DSVS L2
L1 NADA
      CRCT 2
      ARMZ a*
L2 NADA
```

* endereço na pilha D obtido da tabela de símbolos

Novas instruções

Significado	Instrução	Execução
Desvia sempre	**DSVS p**	i:=p;
Desvia se Falso	**DSVF p**	se D[s]=0 **então i:=p** **senão i:=i+1;** s:=s-1;
Comando Nulo	**NADA**	{}

MEPA
Comandos condicionais

- Exemplo: **if (a>b) then q := p and q**

  else if (a <2 *b) then p := true

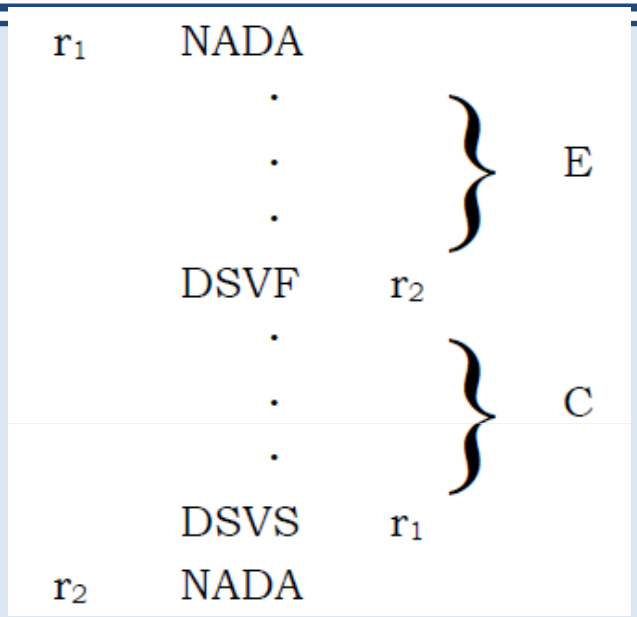
  else q := false

Tradução

```
        CRVL a*          CRVL b*
        CRVL b*          MULT
        CMMA             CMME
        DSVF L3          DSVF L5
        CRVL p*          CRCT 1
        CRVL q*          ARMZ p*
        CONJ             DSVS L6
        ARMZ q*      L5  NADA
        DSVS L4          CRCT 0
L3      NADA             ARMZ q*
        CRVL a*      L6  NADA
        CRCT 2       L4  NADA
```

MEPA
Comandos iterativos

- **while** E **do** C
- Tradução



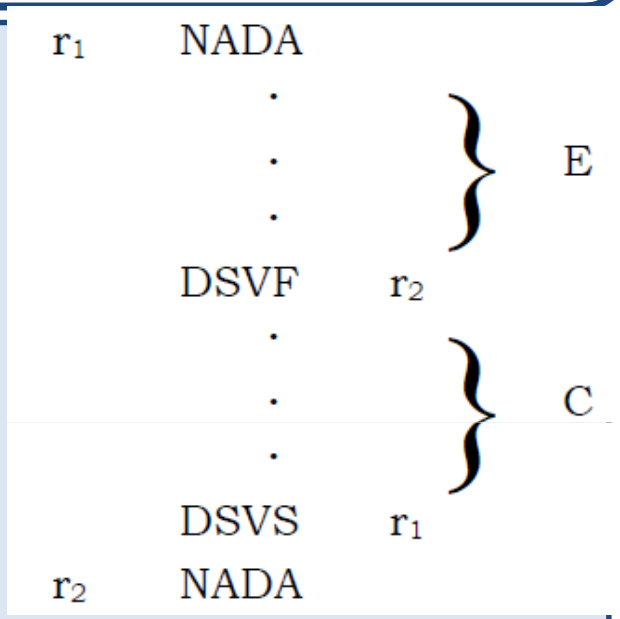
Novas instruções

Significado	Instrução	Execução
Desvia sempre	**DSVS p**	i:=p;
Desvia se Falso	**DSVF p**	se D[s]=0 **então i:=p** **senão i:=i+1;** s:=s-1;
Comando Nulo	**NADA**	{}

MEPA
Comandos iterativos

- **while (s <=n) do s := s +3 * s**

```
L1   NADA          CRVL s*
     CRVL s*       MULT
     CRVL n*       SOMA
     CMEG          ARMZ s*
     DSVF L2       DSVS L1
     CRVL s*  L2   NADA
     CRCT 3
```



Novas instruções

Significado	Instrução	Execução
Desvia sempre	**DSVS p**	i:=p;
Desvia se Falso	**DSVF p**	se D[s]=0 **então i:=p** **senão i:=i+1;** s:=s-1;
Comando Nulo	**NADA**	{}

MEPA
Comandos de entrada/saída

- **read (V1, V2, . . . , Vn)**
- Tradução

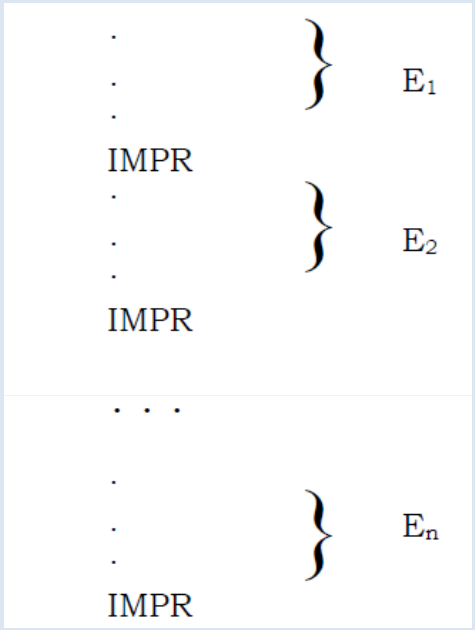
```
LEIT
ARMZ v1*
LEIT
ARMZ v2*
...
LEIT
ARMZ vn*
```

Novas instruções

Significado	Instrução	Execução
Leitura de numero inteiro	LEIT	s:=s+1; D[s] := <entrada padrão>
Leitura de caractere	LECH	s:=s+1; D[s] := <entrada padrão>
Imprime número inteiro	IMPR	<saída padrão> := D[s]; s:=s-1;
Imprime caractere	IMPC	<saída padrão> := D[s]; s:=s-1;
Imprime <enter>	IMPE	

MEPA
Comandos de entrada/saída

- **write (V1, V2, . . . , Vn)**
- Tradução



Significado	Instrução	Execução
Leitura de numero inteiro	LEIT	s:=s+1; D[s] := <entrada padrão>
Leitura de caractere	LECH	s:=s+1; D[s] := <entrada padrão>
Imprime número inteiro	IMPR	<saída padrão> := D[s]; s:=s-1;
Imprime caractere	IMPC	<saída padrão> := D[s]; s:=s-1;
Imprime <enter>	IMPE	

MEPA
Tradução de programas

- Para traduzir um programa simples é necessário:
  - alocar e liberar variáveis;
  - iniciar máquina virtual;
  - terminar máquina virtual.

MEPA – Tradução de programas
Exemplo

Entrada

```
program
• • •
var x, y: integer
• • •
end.
```

Tradução

```
INPP
• • •
AMEM 1
AMEM 1
• • •
DMEM 1
DMEM 1
PARA
```

Cadeia	Token	....	End. relativo
x	ID		0
Y	ID		1

Novas instruções

Significado	Instrução	Execução
Inicia Programa Principal	**INPP**	s:= -1;
Aloca Memória na Pilha, no caso de LALG m=1	**AMEM m**	s := s+m;
Desaloca Memória	**DMEM m**	s := s-m;
Termina a execução	**PARA**	{ }

MEPA
Tradução de programa
Exemplo completo

Entrada

```
program exemplo_1;
var x, y : integer;
begin
        read(x,y);
        if x > y
            then writeln(x)
end.
```

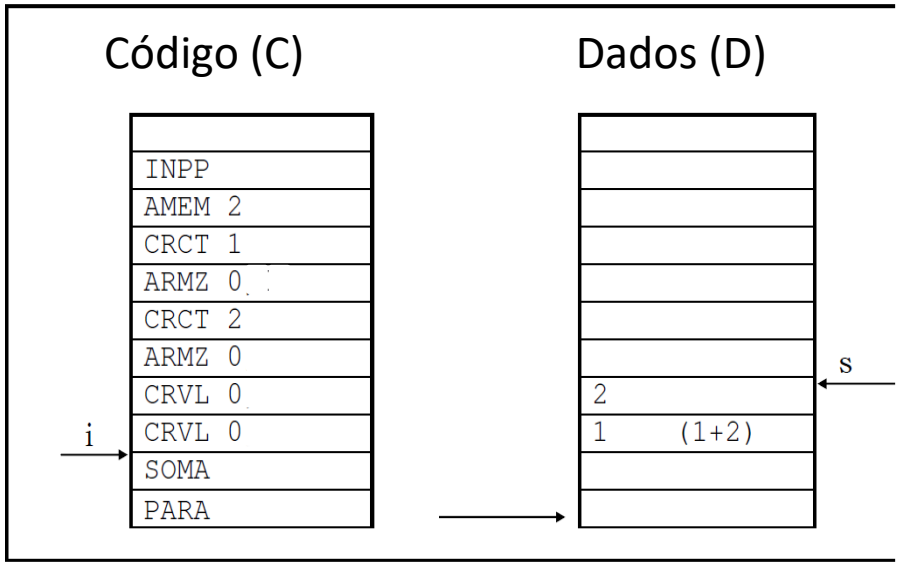
Tradução

```
        INPP
        AMEM    1
        AMEM    1
        LEIT
        ARMZ    0
        LEIT
        ARMZ    1
        CRVL    0
        CRVL    1
        CMMA
        DSVF    0
        CRVL    0
        IMPR
0       NADA
        PARA
```

Geração de código para LALG

- Só será gerado o código de montagem para programas escritos em LALG sem procedimentos.

Detalhes de implementação

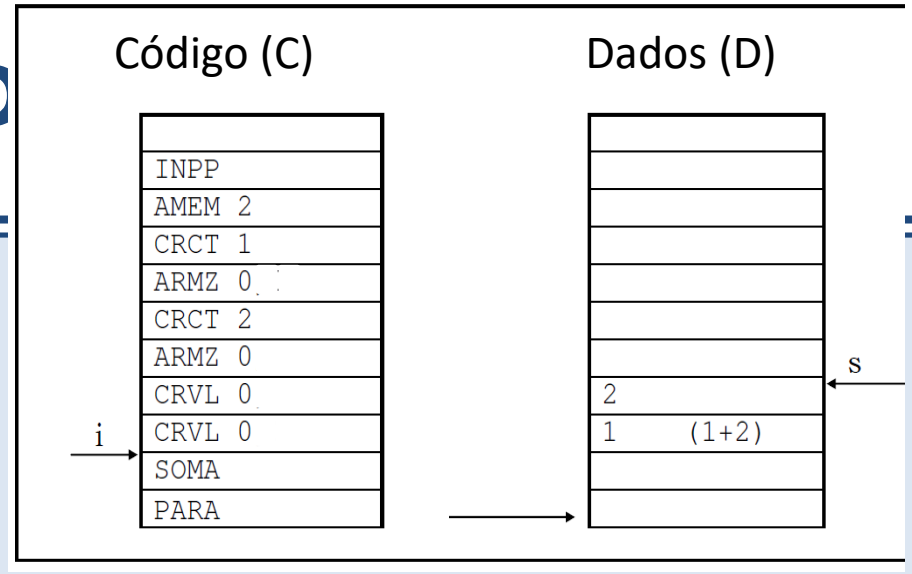


- Programa – vetor C
- Dados – vetor D
- i → Registrador de C e
  s → registrador de D

- Obs:

1. Uma vez que o programa da MEPA está carregado na região C e os registradores têm seus valores iniciais, o funcionamento da máquina é muito simples. As instruções indicadas pelo registrador i são executadas até que seja encontrada a instrução de parada, ou ocorra algum erro. A execução de cada instrução aumenta de 1 o valor de i, exceto as instruções que envolvem desvios.
2. a nível de projeto, o Programa – vetor de código (C) armazena o conjunto de instruções MEPA geradas pelo compilador. Esta saída será a entrada de um programa interpretador deste código.

D entação

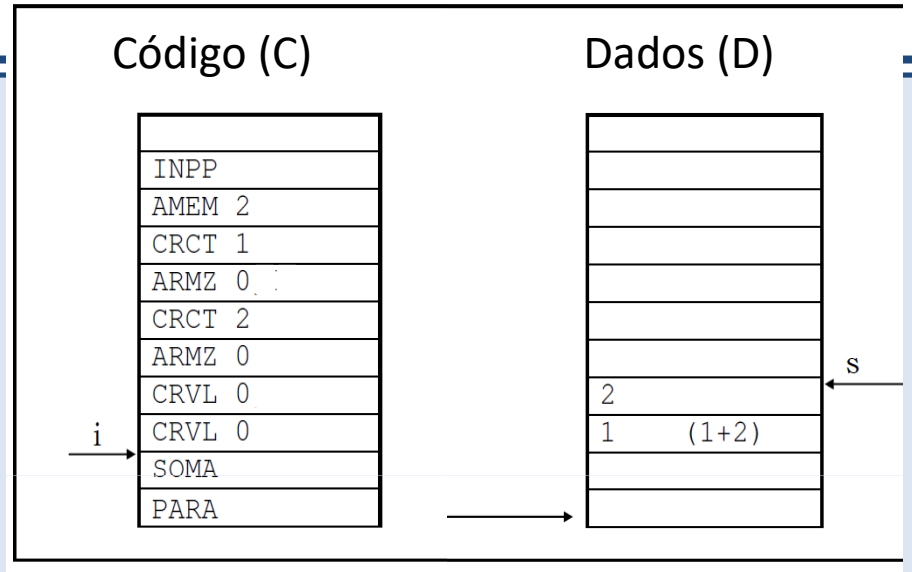


- Como a área D em LALG só manipula inteiros, então

```
var
D: array [0 .. tpilha] of integer;
s: -1 .. tpilha;
```

- A Tabela de Símbolos deverá ser aumentada com um campo: **end_rel** (endereço relativo à base na pilha M).

Detalhes de implementação



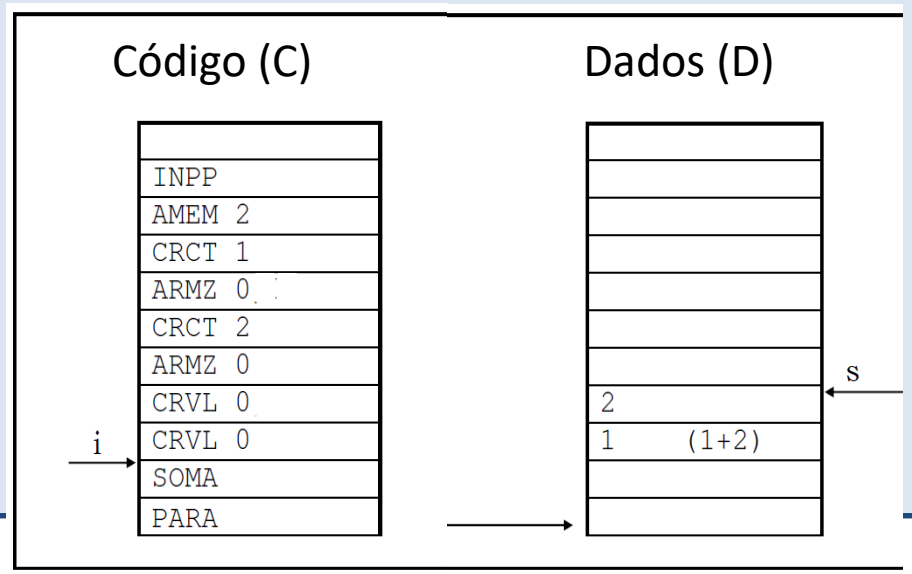

```
Type Funções = (SOMA, SUBT, ARMZ,...);
var C: array[0..MaxC] of
      record
            F: Funções;
            C: -1..MaxInt;
      end;
```

```
procedure Gerar (X: Funções, Y: integer);

begin
      i := i+1;
      with C[i] do
      begin
            F:=X; {código}
            C:=Y; {argumento}
      end
end;
```

Interpretador para LALG

- A entrada para o Interpretador é o array de códigos C. A pilha D será sua área de trabalho e eventualmente ele necessitará de informações da Tabela de Símbolos.
- O procedimento principal seria aquele que percorreria o array C a partir da posição 0, interpretando cada instrução. Terminaria ao encontrar a instrução **PARA**.



Geração de código para LALG

- Definir uma rotina

Gerar(rótulo, código, par1, par2)

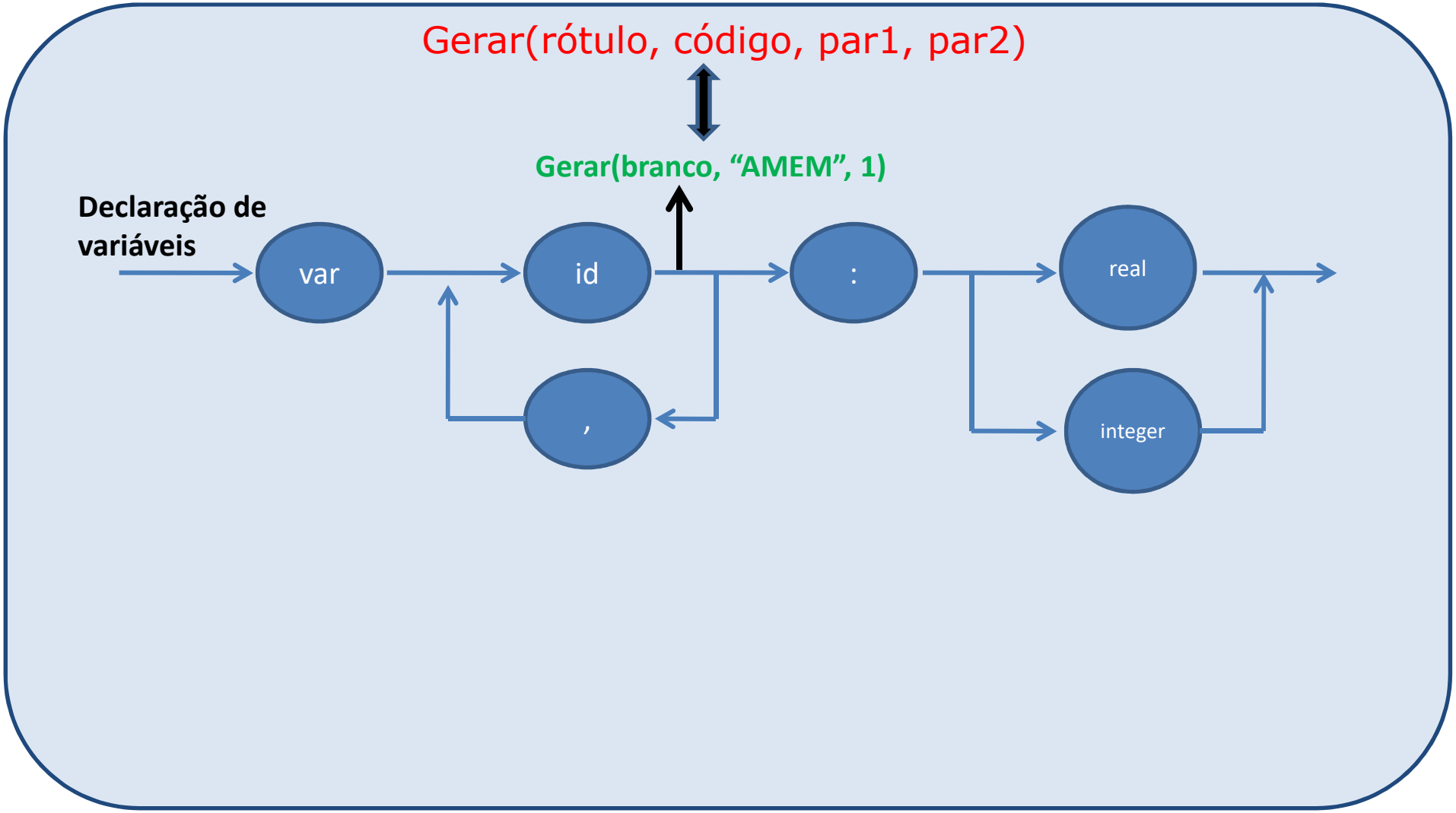
- Que receberá como parâmetros:

rótulo/nada, **código**, seguido de 0,1 ou 2 argumentos

Em caso de desvios

Uma instrução da MEPA

Geração de código para LALG



Rotina Declaração de Variáveis

```
<DC_V> ::= var <VARIAVEIS> : <TIPO_VAR> ; <DC_V> | λ
<TIPO_VAR> ::= integer | real
<VARIAVEIS> ::= <ID> <MAIS_VAR>
<MAIS_VAR> ::= , <VARIAVEIS> | λ
```

```
procedimento dc_v(S)
begin
    se (simb=var) então obter_símbolo()
    senão
        imprimir("Erro: var esperado");
        ERRO(Primeiro(variaveis)+S); //consome até encontrar ID
    variaveis({:}+S);
    se (simb=simb_dp) então obter_símbolo()
    senão
        imprimir("Erro: ':' esperado");
        ERRO(Primeiro(tipo_var)+S); //consome até encontrar integer ou real
    tipo_var({;}+S);
    se (simb=simb_pv) então obter_símbolo()
    senão
        imprimir("Erro: ';' esperado");
        ERRO(Primeiro(dc_v)+S); //consome até encontrar ;
    dc_v(S);
end;
```

Rotina Declaração de Variáveis

```
<DC_V> ::= var <VARIAVEIS> : <TIPO_VAR> ; <DC_V> | λ
<TIPO_VAR> ::= integer | real
<VARIAVEIS> ::= <ID> <MAIS_VAR>
<MAIS_VAR> ::= , <VARIAVEIS> | λ
```

```
procedimento variaveis(S)
begin
s:=0
se (simb=id)
    então
    se busca(cadeia, token="id", cat="var")=false
       então inserir(cadeia,token="id",cat="var", end_rel=s++);Gerar(branco,"AMEM", 1);
       senão ERRO("identificador já declarado");
    obtem_simbolo(cadeia,simbolo)
    enquanto (simbolo=simb_virgula) faça
            obtem_simbolo(cadeia,simbolo)
            se (simb=id)
            então
                se busca(cadeia, token="id", cat="var")=false
     então inserir(cadeia,token="id",cat="var", end_rel=s++); Gerar(branco,"AMEM", 1);
                          senão ERRO("identificador já declarado")
                       obtem_simbolo(cadeia,simbolo)
            fim-então
            senão ERRO(S+{simb_virgula,simb_dois_pontos});
    fim-enquanto
end;
```

Analisador semântico

Geração de código

Rotina Declaração de Variáveis

```
<DC_V> ::= var <VARIAVEIS> : <TIPO_VAR> ; <DC_V> | λ
<TIPO_VAR> ::= integer | real
<VARIAVEIS> ::= <ID> <MAIS_VAR>
<MAIS_VAR> ::= , <VARIAVEIS> | λ
```

```
procedimento tipo_var(S)
begin
se (simb=integer)
    então tab_simb_alterar_tipo(cadeia, token="id", cat="var", tipo="integer")
    senão tab_simb_alterar_tipo(cadeia, token="id", cat="var", tipo="real")
end;
```

var x,y: integer

Cadeia	*Token*	Categoria	Tipo	Valor	...	End.relativo
meu_prog	id	meu_prog	-	-	...	
x	id	var	integer		...	0
y	id	var	integer		...	1

Analisador semântico

Geração de código

```
procedure termo (var t: string);
var t1, t2: string;
begin
Fator (t);
Enquanto simbolo in [*,div, and] faça
begin
    s1:= simbolo;
    simbolo:= analex(s);
    Fator(t1);
    Caso s1 seja
        * : t2 := 'inteiro'; Gerar(branco, "MULT");
        div: t2 := 'inteiro'; Gerar(branco, "DIVI");
        and : t2 := 'booleano' Gerar(branco, "CONJ");
    end;
    Se (t <> t1) ou (t <> t2) então erro('incompatibilidade de tipos')
    end
end;
```

Rotina Termo

```
21. <termo> ::=  <fator>
        {(*| div| and)<fator>}
22.<fator> ::= <variavel>
               | <número>
               | (<expressão>)
               | not <fator>
```

Analisador semântico

Geração de código

```
procedure fator (var t: string);
Inicio
Caso simbolo seja
Número: {t:=inteiro; Gerar(branco, "CRCT", simbolo); simbolo := analex(s);}
Identificador: {Busca(Tab: TS; id: string; ref: Pont_entrada; declarado: boolean);
 Se declarado = false então erro;
  Obtem_atributos(ref: Pont_entrada; AT: atributos);
  Caso AT.categoria seja
      Variavel: {t:= AT.tipo; Gerar(branco, "CRVL", AT.escopo, AT.end); simbolo := analex(s);}
     Constante: {t:= "boolean";
       caso simbolo seja
              "true":  Gerar(branco,"CRCT",1);
              "false": Gerar(branco,"CRCT",0);
       end; simbolo:= analex(s);}
    Else erro;
 fim caso;
Cod_abre_par: {simbolo := analex(s); expressao(t); se simbolo <>
                        Cod_fecha_par  then erro; simbolo := analex(s);}
Cod_neg: {simbolo := analex(s); fator(t);  se t <> 'booleano' então erro
Gerar(branco,"NEGA")
Else erro;
Fim caso;
```

Rotina Fator

```
22.<fator> ::=
  <variavel>
| <número>
| (<expressão>)
| not <fator>
```

Analisador semântico

Geração de código

Geração de código para LALG

- Demais comandos: atribuição, condicional, iterativo, entrada/saída consultar páginas 165-172 do livro "Implementação de Linguagens de Programação" – Tomasz Kowaltowski – Unicamp

- Acrescente ao seu projeto uma rotina para geração de código intermediário e uma rotina para interpretação do código. Obs: código-fonte sem procedimentos
  - **Data limite de entrega: 24/11/18**

Exemplo de geração de código para LALG

- Gere código para o programa abaixo e interprete o código gerado

```
program exemplo_1;
var x, y : integer;
begin
   read(x,y);
   if x > y
            then writeln(x)
end.
```

```
      INPP
      AMEM   1
      AMEM   1
      LEIT
      ARMZ   0
      LEIT
      ARMZ   1
      CRVL   0
      CRVL   1
      CMMA
      DSVF   0
      CRVL   0
      IMPE
0     NADA
      PARA
```

```
Entre com um inteiro:
89
Entre com um inteiro:
2
89
Execução encerrada com sucesso
```

Exemplo de geração de código para LALG

```
program testebytecode;
var x, y, z : integer;
begin
        read(y, z);
        x := x + z*(10 div y + 1);
        writeln(x, y, z);

        if x > 0 then
                if y > 0 then
                    y := x*y
                else
                    y := y + x
        else
                if y > 0 then
                    y := -y
                else
                    y := y + z
end.
```

```
INPP
AMEM 1
AMEM 1
AMEM 1
LEIT
ARMZ 1
LEIT
ARMZ 2
CRVL 0
CRVL 2
CRCT 10
CRVL 1
DIVI
CRCT 1
SOMA
MULT
SOMA
ARMZ 0
CRVL 0
IMPE
CRVL 1
IMPE
```

```
CRVL 2
IMPE
CRVL 0
CRCT 0
CMMA
DSVF 0
CRVL 1
CRCT 0
CMMA
DSVF 1
CRVL 0
CRVL 1
MULT
ARMZ 1
DSVS 2
1 NADA
CRVL 1
CRVL 0
SOMA
ARMZ 1
2 NADA
DSVS 3
```

```
0 NADA
  CRVL 1
  CRCT 0
  CMMA
  DSVF 4
  CRVL 1
  CRCT -1
  MULT
  ARMZ 1
  DSVS 5
4 NADA
  CRVL 1
  CRVL 2
  SOMA
  ARMZ 1
5 NADA
3 NADA
  PARA
```

```
Entre com um inteiro:
2
Entre com um inteiro:
5
30
2
5
Execução encerrada com sucesso
```