Implementação de Mapeamento Finito (*Array*) Dinâmico no Ambiente FURBOL

Acadêmico

Stephan Dieter Bieging

Orientador

José Roque Voltolini da Silva

Roteiro

- Introdução
- Objetivos
- Fundamentação teórica
- Desenvolvimento
- Implementação
- Conclusões
- Extensões

Introdução

- Origem em 1987 “Execução Controlada de Programas”
- Vários TCCs deram continuidade
- Em 1996 passou a se chamar FURBOL e passou a usar Gramática de Atributos para representar a semântica
- Em 1997 foram implementados vários recursos novos no ambiente (chamadas de procedimento, recursividade)

Introdução

- Métodos de passagem de parâmetros e produto cartesiano foram implementados em 1999 usando a notação BNF e gramática de atributos
- Geração de código executável foi implementado em 2000
- Em 2001 o ambiente foi estendido através da implementação de mapeamento finito (*array*)
- É utilizada a especificação apresentada por Adriano (2001)

Introdução

- Motivação
  - Conhecimento sobre compiladores
  - A continuação de um trabalho que vem sendo desenvolvido através de trabalhos de conclusão de curso

Objetivos

- Implementação de *arrays* dinâmicos
  - permitindo a utilização de *arrays* onde os limites são variáveis e conhecidos somente em tempo de execução
- Acesso a variáveis não-locais
  - permitindo que procedimentos aninhados acessem variáveis declaradas em procedimentos mais envolventes

Fundamentação Teórica

- Compiladores
  - Análise léxica
  - Análise sintática
  - Análise semântica
- Conceito de amarração
  - Entidade variável
    - Escopo, tempo de vida, valor e tipo
  - Unidades de programas
    - Segmento de código
    - Registro de ativação

Fundamentação Teórica (cont.)

- Alocação memória
  - Linguagens estáticas
    - Organização de memória
  - Linguagens dinâmicas
    - Alocação implícita
    - Alocação explícita
    - Organização de memória
      - Memória de pilha
      - Memória *heap*

Fundamentação Teórica (cont.)

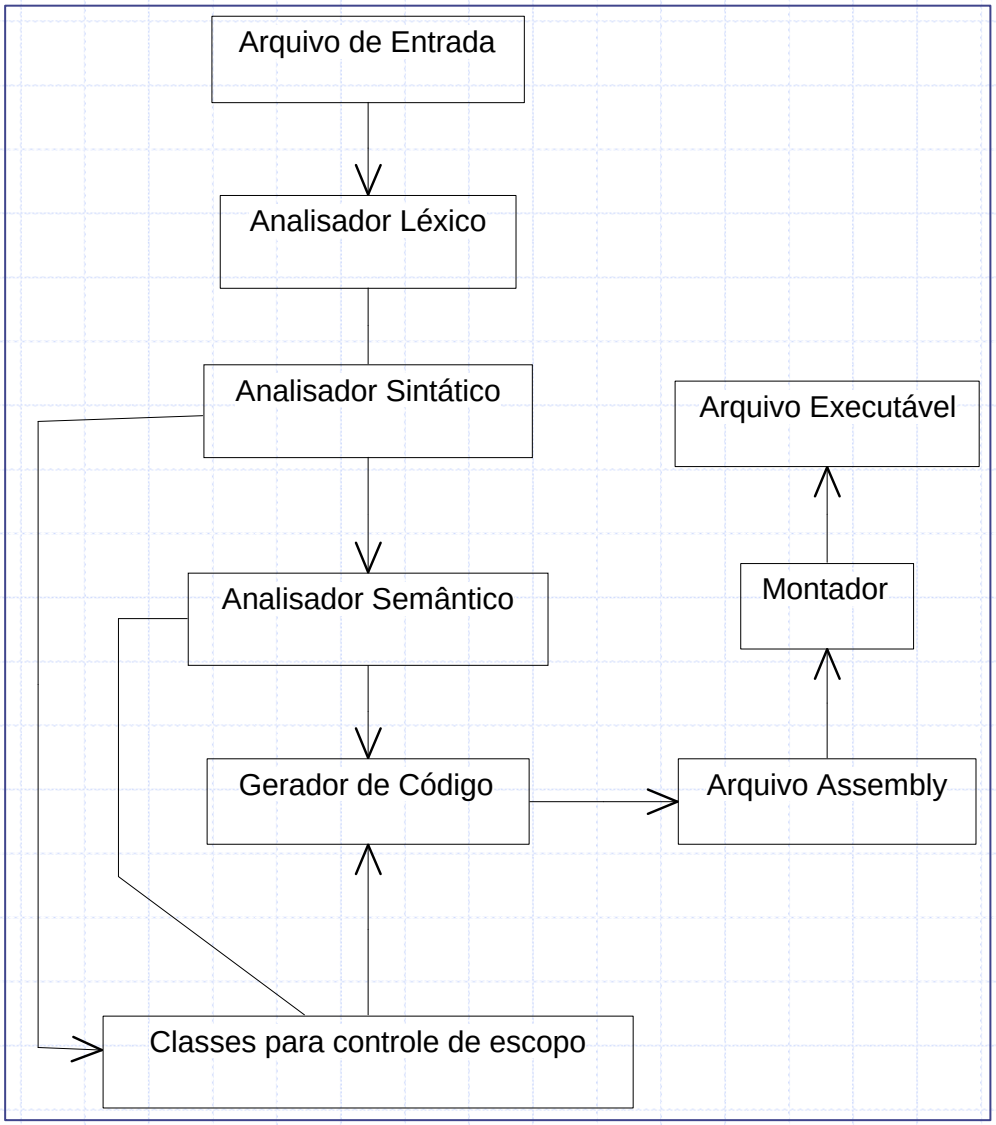
- Mapeamento finito (*array*)
  - *Arrays* e índices
  - Implementação (estudos sobre as fórmulas para acesso aos elementos do *array*)
- Acesso à variáveis não-locais
  - Escopo estático com procedimentos aninhados
  - Regra do aninhamento mais interno
  - *Elos de acesso*

Fundamentação Teórica (cont.)

- Geração de código intermediário
  - Código de três endereços ($x := y$ op z)
- Geração de código-alvo
- Linguagem *Assembly*
  - Linguagem de computador singular
  - Instruções com 3 ou 4 caracteres
- Serviços do DOS
  - Serviços de arquivo/diretório, caractere, gerenciamento de memória etc.

Apresentação Geral

- Setas indicam o caminho da informação
- Relações sem setas são bidirecionais



Acesso à Variáveis Não-Locais

- Escopo estático com procedimentos aninhados
- Regra do aninhamento mais interno
- Implementação através de *elos de acesso*
  - O registro de ativação contém um ponteiro para o registro de ativação do pai estático
  - A cada chamada de procedimento empilham-se os parâmetros (se existirem) e a base do registro do ativação do pai estático do procedimento chamado

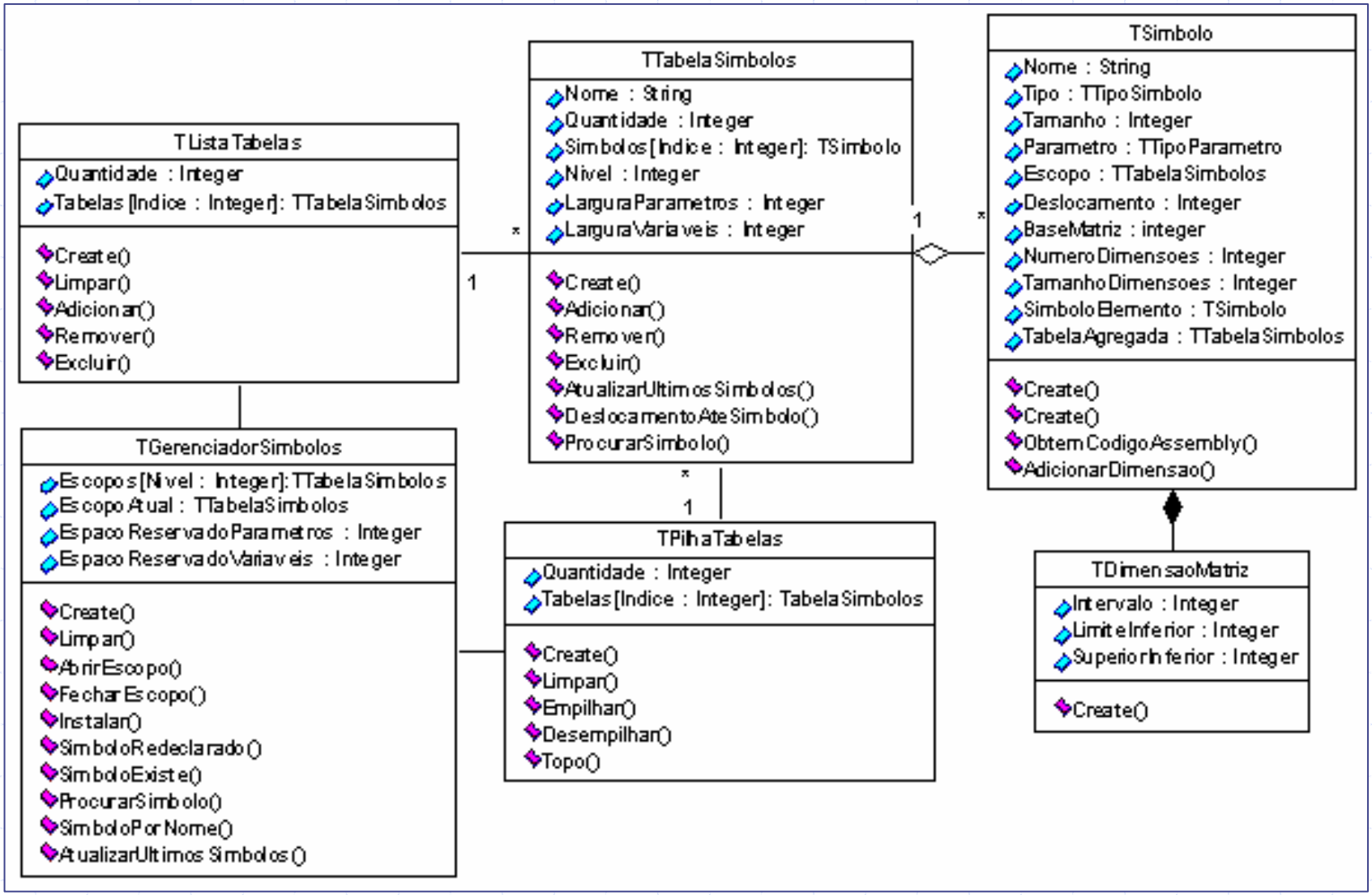
Exemplo de Acesso Não-Local

```
programa Escopo;
  procedimento P;
  var
    VarLocalP: Inteiro;

    procedimento Q;
    var
      VarLocalQ: Inteiro;
    inicio
      VarLocalQ := VarLocalP;
    fim;
  inicio
    Q;
  fim;

inicio
  P;
fim.
```

```
   ; DI tem o elo de acesso
MOV     DI,[BP+4]
   ; É usado o deslocamento
MOV     AX,WORD PTR [DI-2]
    ; variável no registro
MOV     WORD PTR [BP-2],AX
```

```
   ; Empilhar o elo de acesso
PUSH    BP
   ; Chamada do proc. Q por P
CALL    Q
```

Especificação: Classes de Escopo



Mapeamento Finito Dinâmico

- Implementação *array* dinâmico
  - Necessário a liberação de memória alocada pelo DOS
  - Descritor contém informações sobre o *array*
    - Ponteiro para área de dados dos elementos, número de dimensões, endereço relativo ao primeiro elemento do *array*, limites inferior e superior de cada dimensão
  - Construções implementadas
    - Declaração, comando *redimensiona* e as funções *dimensoes*, *liminf*, *limsup* e o acesso aos elementos do *array*

Construções para *Arrays* Dinâmicos

- Declaração de *arrays* dinâmicos
  - Internamente é um ponteiro
  - Inicializado com zero
- Comando *Redimensiona*
  - Avaliação das expressões
  - Alocação e inicialização do descritor
  - Cálculo do número de elementos
  - Alocação da área de dados
  - Cálculo do endereço relativo ao primeiro elemento do *array*

Fórmulas usadas na alocação de *Arrays* Dinâmicos

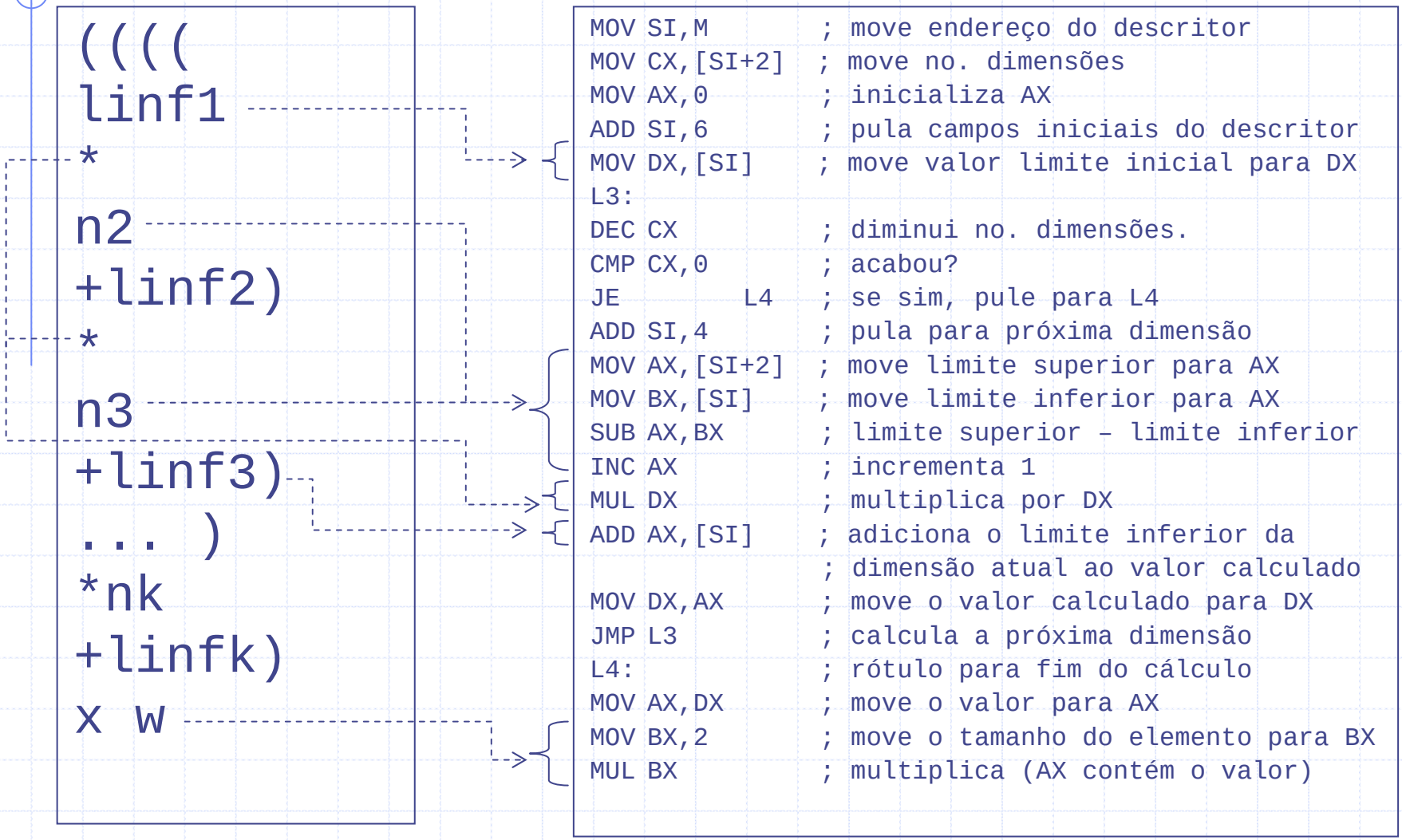
Fórmula genérica para cálculo do elemento de um *array*

```
(( ... ((i1n2 + i2)n3 + i3) ... )nk + ik)
x w + base – (( ... (( linf1n2 + linf2 )n3
        + linf3) ... )nk + linfk) x w
```

Fórmula para o endereço relativo ao primeiro elemento

```
((((linf1n2 + linf2)n3 + linf3) ... )nk +
                    linfk) x w
```

Código Gerado pela Fórmula de Cálculo do Endereço Relativo do Primeiro Elemento



Construções para *Arrays* Dinâmicos

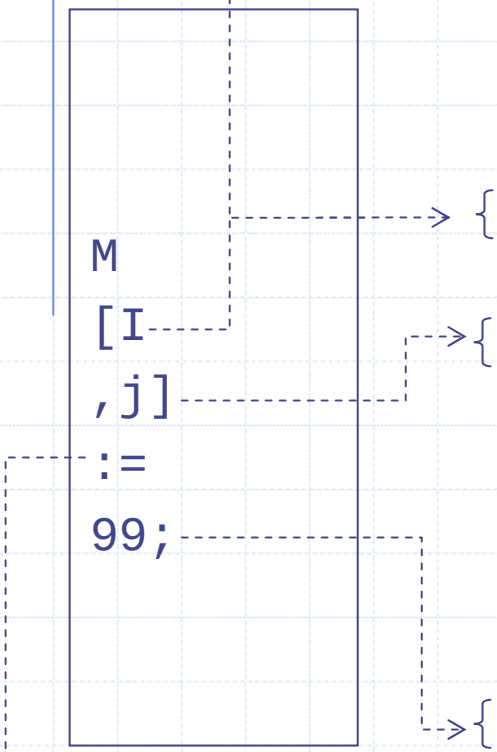
- Função *dimensoes*
  - O retorno desta função é obtido do descritor do *array*
- Função *liminf* e *limsup*
  - Parâmetro indica a dimensão
  - Valor proveniente do descritor
- Acesso aos elementos
  - Avaliação das expressões
  - Cálculo do endereço do elemento
- Liberação de *array* dinâmico

Acesso aos Elementos do *Array*

```
programa AcessoElementos;
var
  M: matriz: Inteiro;
  i, j: inteiro;
inicio
  ...
  i := 2; j := 10;
  M[i,j] := 99;
fim.
```

$$\frac{((\ldots((i1n2 + i2)n3 + i3)\ldots)nk + ik)}{x\ w + base} - ((\ldots((linf1n2 + linf2)n3 + linf3)\ldots)nk + linfk)\ x\ w$$

Código Gerado pelo Acesso ao elemento de um *Array* Dinâmico




```
MOV SI,I        ; move o valor de I para SI
MOV DI,M        ; obtém o descritor do array
ADD DI,6+4      ; pula os campos iniciais e a 1° dim.
MOV AX,[DI+2]   ; AX = limite inferior 2° dimensão
SUB AX,[DI]     ; AX = AX – limite superior 2° dim.
INC AX          ; soma 1
MOV DI,AX
MOV AX,DI
MOV DI,SI       ; move o valor de I (SI) para DI
MUL DI          ; mult. pelo intervalo da 2° dim.
MOV DI,AX       ; move para DI
ADD DI,J        ; adiciona o valor de J
MOV DI,DI
MOV AX,2        ; move o tamanho do elemento
MUL DI          ; no. de dimensões * tamanho do elemento
MOV DI,AX       ; move o valor obtido para DI
MOV SI,M        ; obtém o descritor
SUB DI,[SI+4]   ; subtrai DI do endereço do 1° elem.
MOV BX,DI       ; move o valor para BX
PUSH BX         ; e empilha
MOV AX,99       ; move o valor 99 par AX
POP DI          ; desempilha end. relat. do elem.
MOV SI,M        ; obtém o descritor do array
MOV SI,[SI]     ; obtém end. da área de dados array
ADD SI,DI       ; adiciona deslocamento ao elemento
MOV [SI],AX     ; move o valor para o elemento
```

Especificação

Programa	::=	'PROGRAMA'	vSimbolos.Limpar; GeraAsm('CODIGO SEGMENT'); GeraAsm('ASSUME CS:CODIGO, DS:CODIGO'); GeraAsm('ORG 100H');

		#ID	ID.nome := #ID;
			GeraAsm('ENTRADA: JMP' || #ID.nome);

		';'	vSimbolos.AbrirEscopo('Principal');
		EstruturaDados	
		EstruturaSubRotinas	
		Ccomposto	GeraAsm(EstruturaSubRotinas.CodAsm ||ID.nome || 'PROC NEAR' || 'PUSH BP' ||'MOV BP,SP' 'MOV AX,CS' || 'MOV ES,AX' || 'MOV BX,4096' || 'MOV AH,4AH' || 'INT 21H' || CodIni*Arrays*(vSimbolos.EscopoAtual) || CComposto.CodAsm || CodLiberar*Arrays*(vSimbolos.EscopoAtual) || 'POP BP' || 'INT 20H' || ID.nome || 'ENDP');
		'.'	DeclaraDadosGlobais; GerarAsm('ultimo_rotulo:'); GerarAsm('CODIGO ENDS); GerarAsm('END ENTRADA');

Especificação (cont.)

Bloco	::=	EstruturaDados	
		EstruturaSubrotina	
		Ccomposto	Bloco.Codigo := EstruturaSubrotina.Codigo || CComposto.Codigo; Bloco.CodAsm := EstruturaSubRotina.CodAsm || CComposto.CodAsm;
CComposto	::=	'INICIO'	
		Comando	CComposto.Codigo := Comando.Codigo; CComposto.CodAsm := Comando.CodAsm;
		'FIM'	

EstruturaDados	::=	'VAR'	
		#ID	Se vSimbolos.SimboloRedeclarado(ID) então Erro('Identificador redeclarado'); vSimbolos.Instalar(Tsimbolo.Create(#ID.nome));
		ListaID	
		(Matriz	
		| ^)	vSimbolos.AtualizarUltimosSimbolos(NroVar, ListaId.Tipo);
		';'	
		Declarações	
Declaracoes	::=	#ID	Se vSimbolos.SimboloRedeclarado(ID) então Erro('Identificador redeclarado'); vSimbolos.Instalar(TSimbolo.Create(ID.nome));
		ListaID	
		(Matriz	
		| ^)	vSimbolos.AtualizarUltimosSimbolos(NroVar, ListaId.Tipo);
		';'	
		Declaracoes	

Especificação (cont.)

ListaID	::=	','	
		ID	Se vSimbolos.SimboloRedeclarado(ID) entao Erro('Símbolo redeclarado'); vSimbolos.Instalar(TSimbolo.Create(ID.nome));
		ListaID	
		(Matriz	
		| ^)	vSimbolos.AtualizarUltimosSimbolos(NroVar, ListaId.Tipo); NroVar := 0;
	|	':'	
		Tipo	ListaId.Tipo := Tipo.Tipo;
Tipo	::=	'INTEIRO'	Tipo.Tipo := tsInteiro;
	|	'LOGICO'	Tipo.Tipo := tsLogico;
	|	'MATRIZ'	Tipo.Tipo := tsMatriz;

Especificação (cont.)

```
m: matriz: inteiro;
```

Matriz	::=	LimitesM
		':'
		Tipo
			Simb:=
			  vSimbolos.ProcurarSimbolo(Matriz.NomeId);
			se Simb.NumeroDimensoes > 0 entao
			  Tipo := tsMatriz
			senao
			  Tipo := tsMatrizDinamica;
			vSimbolos.AtualizarUltimosSimbolos(NroVar,
			  Tipo.Tipo);
			NroVar := 0;

	|	^	
LimitesM	**::=**	**'['** **Dimensao** **']'**	
	|	**^**	
Dimensao	::=	#NUM_1 '..' #NUM_2	Simb:=vSimbolos.SimboloPorNome(Dimensao.NomeId); Simb.AdicionarDimensao(NUM_1.valor, NUM_2.valor);
		MaisDimensao	
MaisDimensao	::=	',' Dimensao	
	|	^	

Especificação (cont.)

EstruturaSubRotina	::=	EstruturaProcedimento	EstruturaSubRotina.Codigo := EstruturaProcedimento.Codigo EstruturaSubRotina.CodAsm := EstruturaProcedimento.CodAsm;
	|	^	
EstruturaProcedimento	::=	'PROCEDIMENTO'	
		#ID	Se vSimbolos.SimboloRedeclarado(ID.nome) entao Erro('Símbolo redeclarado'); Simb:=TSimbolo.Create(ProcName, tsProcedimento); vSimbolos.Instalar(Simb); Simb.TabelaAgregada := vSimbolos.AbrirEscopo(ID.nome);
		ParamFormais	
		';'	
		EstruturaDados	
		EstruturaSubRotina	
		CComposto	EstruturaProcedimento.CodAsm := ID.nome || 'PROC NEAR' || 'PUSH BP' || 'MOV BP,SP' || 'SUB SP,' || vSimbolos. EscopoAtual.LarguraVariaveis || CComposto.CodAsm || 'MOV SP,BP' || 'POP BP' || 'RET' vSimbolos. EscopoAtual.LarguraParametros + 2 || ID.nome || 'ENDP' || EstruturaSubRotinas.CodAsm;
		';'	vSimbolos.FecharEscopo;
		EstruturaSubRotinas	EstruturaProcedimento.Codigo := EstruturaProcedimento.Codigo || EstruturaSubRotina.Codigo; EstruturaProcedimento.CodAsm := EstruturaProcedimento.CodAsm || EstruturaSubRotina.CodAsm;

Especificação (cont.)

ParamFormais	::=	'('	
		(ParamValor |	
		ParamRef)	
		SecaoParam	
		')'	
	|	^	
SecaoParam	::=	';'	
		(ParamValor |	
		ParamRef)	
		SecaoParam	
	|	^	
ParamValor	::=	#ID	Inc(NroVar); Se vSimbolos.SimboloRedeclarado(ID.nome) entao Erro('Símbolo redeclarado'); Simb:=TSimbolo.Create(ID.nome); Simb.Parametro := tpValor; vSimbolos.Instalar(Simb);
		ListaID	vSimbolos. AtualizarUltimosSimbolos(NroVar,ListaId.Tipo); NroVar := 0;
ParamRef	::=	'REF'	
		#ID	Inc(NroVar); Se vSimbolos.SimboloRedeclarado(ID.nome) entao Erro('Símbolo redeclarado'); Simb:=TSimbolo.Create(ID.nome); Simb.Parametro := tpReferencia vSimbolos.Instalar(Simb);
		ListaID	vSimbolos. AtualizarUltimosSimbolos(NroVar,ListaId.Tipo); NroVar := 0;

Especificação (cont.)

Comando	::=	#ID	se nao vSimbolos.ProcurarSimbolo(ID) entao Erro('Símbolo não declarado');
		Atribuição	Comando.Codigo:=Atribuição.Codigo; Comando.CodAsm:=Atribuição.CodAsm;

	|	ChamProc	NivelChamador := vSimbolos.EscopoAtual.Nivel;
			NivelChamado := ID.Nivel;
			se NivelChamador<NivelChamado entao
			  CodigoEloAcesso := 'PUSH BP'
			senao
			  CodigoEloAcesso := GerarEloAcesso(NivelChamador
			  - NivelChamado + 1) || 'PUSH DI';
			Comando.Codigo := Comando.Codigo || ChamProc.Codigo
			  || CodigoEloAcesso || 'CHAMADA' ||ChamProc.Nome;
			Comando.CodAsm := Comando.CodAsm || ChamProc.CodAsm
			  || CodigoEloAcesso || 'CALL' || ChamProc.Nome;

		Virgula	
	|	CCondicional	
		Virgula	Comando.Codigo := CCondicional.Codigo; Comando.CodAsm := CCondicional.CodAsm;
	|	CRepetição	
		Virgula	Comando.Codigo := CRepetição.Codigo; Comando.CodAsm := CRepetição.CodAsm;
	|	CEntrada	
		Virgula	Comando.Codigo := CEntrada.Codigo; Comando.CodAsm := CEntrada.CodAsm;

Especificação (cont.)

Comando	::=	CSaida	
		Virgula	Comando.Codigo := CSaida.Codigo; Comando.CodAsm := Comando.CodAsm;
	|	CInc	Comando.Codigo := CInc.Codigo; Comando.CodAsm := CInc.CodAsm;
		Virgula	
	|	CDec	Comando.Codigo := CDec.Codigo; Comando.CodAsm := CDec.CodAsm;
		Virgula	
	|	'NL'	
		Virgula	Comando.Codigo := CNL.Codigo; Comando.CodAsm := Comando.CodAsm;
	|	CRedimensiona	Comando.Codigo := Credimensiona.Codigo; Comando.CodAsm := Credimensiona.CodAsm;
		Virgula	
	|	Virgula	
Virgula	::=	';'	
		Comando	Virgula.Codigo:=Comando.Codigo Virgula.CodAsm:=Comando.CodAsm
	|	^	

Especificação (cont.)

Atribuicao	::=	':='	se Atribuição.Deslocamento <> '' então
			inicio
			  RX:=novo_t;
			  IndCodAsm := 'MOV RX,' || Ldesloca || 'PUSH RX';
			  PreIdAt := 'POP DI';
			end;

		Expressao	Atribuicao.Local := Expressao.local;
			se Expressao.tipo <> Atribuicao.tipo entao
			  Erro;
			se Expressão.Deslocamento <> '' então
			  Atribuição.Codigo := idAT || '[' ||
			  Expressao.Deslocamento || ']:=' ||
			  Expressao.Local;
			senão
			  Atribuição.Codigo := IdAT || ':=' ||
			  Expressao.Local);
			se Simb.Tipo = tsMatrizDinamica entao
			inicio
			  PreIdAt := PreIdAt || 'MOV SI,[' || IdAt || ']'
			  'MOV SI,[SI] || 'ADD SI,DI';
			  IdAt := '[SI]';
			fim;
			Atribuição.CodAsm := IndCodAsm || Expressao.CodAsm ||
			  AtPrelocal || 'MOV RX,' || aTlocal || PreIdAT ||
			  'MOV' || IdAT || ',' || RX;

Especificação (cont.)

ChamProc	::=	'('	
		ParamAtuais	ChamProc.CodAsm := ParamAtuais.CodAsm;
		')'	
	|	^	

ParamAtuais	::=	#ID	SimbVar := vSimbolos.SimboloPorNome(lexema);
			SimbPar := SimbProc.Simbolos[parN - 1]
			se SimbVar.Tipo <> SimbPar.Tipo entao
			  Erro('Tipos incompatíveis');
			se SimbVar.Parametro = tpReferencia entao
			  ParamAtuais.CodAsm := 'LEA DI,' || ID.nome ||
			  'PUSH DI' || ParamAtuais.CodAsm;
			se SimbVar.Parametro = tpValor entao
			  ParamAtuais.CodAsm := 'PUSH' || ID.nomeAsm ||
			  ParamAtuais.CodAsm;

		(','	
		ParamAtuais	
		| ^)	

Especificação (cont.)

CRepetição	::=	'ENQUANTO'	CRepetição.Inicio := novo_l; Expressao.v := novo_l; Expressao.f := CRepetição.Proximo;
		Expressao	se Expressao.Tipo <> tsLogico entao Erro('Tipos incompatíveis');
		'FACA'	

		CComposto	CComposto.Proximo := CRepetição.inicio;
			CRepetição.Codigo := CRepetição.Inicio ':' ||
			  Expressão.Codigo || E.v || ':' ||
			  CComposto.Codigo || GOTO ||
			  CRepetição.Inicio;
			CRepetição.CodAsm := CRepetição.Inicio ':' ||
			  Expressão.CodAsm || E.v ':' ||
			  CComposto.CodAsm || 'JMP' ||
			  CRepetição.inicio;

Especificação (cont.)

Ccondicional	::=	'SE' Expressao 'ENTAO'	

		CComposto	se Expressao.Tipo <> tsLogico entao
			  Erro('Tipo incompatível');
			CComposto.Proximo:=CCondicional.Proximo;
			CCondicional.Codigo:=Expressão.Codigo ||
			  E.v || ':' || CComposto.Codigo;
			CCondicional.CodAsm:=Expressão.CodAsm ||
			  E.v || ':' || CComposto.CodAsm;

		CCondicional2	CCondicional.codigo:=CCondicional.Codigo
			  || CCondicional2.Codigo;
			CCondicional.CodAsm:=CCondicional.CodAsm
			  || CCondicional2.CodAsm;

CCondicional2	::=	'SENÃO'	

		CComposto	CCondicional2.Codigo := Expressao.Codigo
			  || 'goto' || proximo || CCondicional2.f
			  ':' || CComposto.Codigo;
			CCondicional2.CodAsm := Expressao.CodAsm
			  || 'JMP' || proximo || CCondicional2.f
			  ':' || CComposto.CodAsm;

	|	^	CCondicional2.Codigo :=
			  Ccondicional2.Codigo || f || ':';
			Ccondicional2.CodAsm :=
			  Ccondicional2.Codasm || f || ':';

Especificação (cont.)

CEntrada	::=	'LEITURA'	
		'('	
		#ID	se nao vSimbolos.ProcurarSimbolo(ID.nome) entao Erro('Identificador não declarado');
		LeituraListaId	
		')'	CEntrada.Codigo := 'LEITURA(ID1' || LeituraListaID.Codigo || ')';
LeituraListaID	::=	','	
		#ID	se nao vSimbolos.ProcurarSimbolo(ID.nome) entao Erro('Identificador não declarado'); LeituraListaID.Codigo:= LeituraListaID.Codigo || ',' || ID;
		LeituraListaId	
	|	^	

Especificação (cont.)

Csaida	::=	'IMPRIME'	
		'('	
		Expressao	
		ListaExpressao	
		')'	CSaida.Codigo := 'Imprime' || Expressao.Codigo || ',' || ListaExpressao.Codigo;
ListaExpressao	::=	','	
		Expressao	ListaExpressao.Codigo := ListaExpressao.Codigo || Listaexpressao.Codigo;
		ListaExpressao	
	|	^	

Especificação (cont.)

CInc	::=	'('	

		#ID	se vSimbolos.ProcurarSimbolo(ID.nome) entao
			  Erro('Símbolo não declarado');
			se ID.Tipo <> 'inteiro' entao
			  Erro('Tipos incompatíveis');
			CInc.Codigo := 'INC(' || ID.nome || ')';
			CInc.CodAsm := 'INC' || ID.nome;

		')'	
CDec	::=	'('	

		#ID	se vSimbolos.ProcurarSimbolo(ID.nome) entao
			  Erro('Símbolo não declarado');
			se ID.Tipo <> 'inteiro' entao
			  Erro('Tipos incompatíveis');
			CDec.Codigo := 'DEC(' || ID.nome || ')';
			CDec.CodAsm := 'DEC' || ID.nome;

		')'	

Especificação (cont.)

```
Redimensiona(M, 1..10, 1..10);
```

CRedimensiona	::=	'REDIMENSIONA'	
		'('	

		#ID	Simb:=vSimbolos.
			  ProcurarSimbolo(#ID.nome);
			se nao Simb <> nil entao
			  Erro('Identificador não
			 declarado);
			se Simb.Tipo<>tsMatrizDinamica
			 entao erro('Identificador não é uma matriz
			dinâmica');

		','	

		DimensaoDinamica	RotuloDimensoes:=novo_l;
			RotuloInicializa:=novo_l;
			RotuloComecoCalcula:=novo_l;
			RotuloFimCalcula:=novo_l;
			se (Simb.Escopo.Nivel>0) and
			  (Simb.Escopo.Nivel<
			  vSimbolos.EscopoAtual.
			  Nivel) então
			inicio
			  AcessoVariavelInicio := 'PUSH DI' ||
			  GerarEloAcesso(vSimbolos.
			  EscopoAtual.Nivel-Simb.Escopo.Nivel);
			  AcessoVariavelFim := 'POP DI';
			  RegistradorVariavel:='DI'; fim; senao
			inicio
			  AcessoVariavelInicio:='';
			  AcessoVariavelFim:='';
			  RegistradorVariavel := 'BP'; fim;
			CodAsm := CodigoAsmMatriz;

		')'	

Especificação (cont.)

```
Redimensiona(M, 1..10, 1..10);
```

DimensaoDinamica	::=	Expressao	se Expressão.Tipo<>tsInteiro entao Erro('Tipos incompatíveis'); R := novo_t; DimensaoDinamica.CodAsm:= Expressao.CodAsm || PreLocal || 'MOV' || R || ',' || Local || 'PUSH' || R;
		'..'	
		Expressao	se Expressão.Tipo<>tsInteiro entao Erro('Tipos incompatíveis'); R := novo_t; DimensaoDinamica.CodAsm:= Expressao.CodAsm || PreLocal || 'MOV' || R || ',' || Local || 'PUSH' || R;
		MaisDimensaoDinamica	Dimensão.CodAsm:=Dimensão. CodAsm + MaisDimensaoDinamica.CodAsm;
MaisDimensaoDinamica	::=	','	
		DimensaoDinamica	
	|	^	

Especificação (cont.)

Expressão	::=	Expressão2	Relação.v:=Expressão2.v; Relação.f:=Expressão2.f; Relação.local:=Expressão2.local; Relação.codigo:=Expressão2.codigo; Relação.codAsm:=Expressão2.codAsm;
		Relação	Expressão.local :=Relação.local; Expressão.Codigo:=Relação.Codigo; Expressão.CodAsm:=Relação.CodAsm; Expressão.v:=Relação.v Expressão.f:=Relação.f

Relação	::=	'='	
		Expressao2	Relacao.Codigo := 'SE' || Relação.local '=' Expressão2.Local || 'goto' || E.v || 'goto' || E.f; Relacao.CodAsm := 'MOV R0,' || Relacao.local || 'CMP R0,' || local || 'JE' RV || 'JMP Rf';
	|	'<>'	
		Expressao2	Relacao.Codigo:= 'SE' || Relação.local || '<>' || Expressão2.local || 'goto' || E.v || 'goto' || E.f; Relação.CodAsm := 'MOV R0,' || Relacao.local || 'CMP R0,' || local || 'JNE' || Rv || 'JMP Rf';
	|	'<'	
		Expressao2	Relacao.Codigo := 'SE' || Relação.local '<' Expressão2.local || 'goto' || E.v || 'goto' || E.f; Relacao.CodAsm := 'MOV R0,' || Relacao.local || 'CMP R0,' || local || 'JB' || Rv || 'JMP' Rf;
	|	'>'	
		Expressao2	Relacao.Codigo := 'SE' || Relação.local ' >' Expressão2.local || 'goto' || E.v || 'goto' || E.f; Relacao.CodAsm := 'MOV R0,' || Relacao.local || 'CMP R0,' || local || 'JA' || Rv || 'JMP' || Rf;
	|	^	

Especificação (cont.)

EL	::=	'+'	
		TC	EL_1.Local := novo_t; EL_1.Codigo := EL.Codigo || TC.Codigo || EL_1.local ':=' EL.local '+' || TC.local; EL_1.CodAsm := EL.Codigo|| TC.Codigo || 'MOV RX,' || EL.local || 'ADD RX,' || Tc.Local || 'MOV' || EL_1.local ',RX';
		EL_1	EL.local := EL_1.local; EL.Codigo := EL_1.Codigo; EL.CodAsm := EL_1.CodAsm;
	|	'-'	
		TC	EL_1.local:= novo_t; EL_1.Codigo:= EL.Codigo || TC.Codigo || EL_1.local || ':=' || EL.local || '-' || TC.local; EL_1.CodAsm := EL.Codigo || TC.Codigo || 'MOV 'RX,' || EL.local || 'SUB RX,' || Tc.Local || 'MOV' || EL_1.local || ',RX';
		EL_1	EL.local := EL_1.local; EL.Codigo := EL_1.Codigo; EL.CodAsm := EL_1.CodAsm;
	|	^	
TC	::=	F	TL.v := F.v; TL.f := F.f; TL.local := F.local; TL.Codigo := F.Codigo; TL.CodAsm := F.CodAsm;
		TL	T.v := TL_s.v; T.f := TL_s.f; TC.local := TL.local; TC.CodAsm := TL.CodAsm; TC.Codigo := TL.Codigo;

Especificação (cont.)

TL	::=	'*'	
		F	TL_1.local := novo_t; TL_1.Codigo := TL.Codigo || F.Codigo || TL1.local ':=' || TL.local '*' || F.local; TL_1.CodAsm := TL.Codigo || F.Codigo || 'MOV AX,' || TLLOCAL || 'MUL ' || Local || 'MOV' || TliLocal || ',AX;
		TL_1	TL.local := TL_{1s}.local; TL.Codigo := TL_{1s}.Codigo; TL.CodAsm := TL_{1s}.CodAsm;
	|	'/'	
		F	TL_1.local := novo_t; TL_1.Codigo:=TL.Codigo || F.Codigo || 'TL$_1$.local || ':=' || TL.local || '/' || F.local; TL_1.CodAsm := TL.Codigo || F.Codigo || 'MOV AX,' || TLLOCAL || 'DIV' || Local || 'MOV' || TliLocal || ',AX;
		TL_1	TL.local := TL_1.local; TL.Codigo := TL_s.Codigo; TL.CodAsm := TL_1.CodAsm;
	|	'E'	
		F	TL.v := novo_l; TL.f := TL_1.f; F.v := TL_1.v; F.f :=TL_1.f;
			TL_1.Codigo := TL.Codigo || TL_1.v || ':' || F.Codigo; TL_1.CodAsm := TL.CodAsm || TL_1.v || ':' || F.CodAsm;
		TL_1	$TL.v := TL_1.v$; $TL.f := TL_1.f$; TL.CodAsm := TL_{1s}.CodAsm; TL.Codigo := TL_{1s}.Codigo;
	|	^	

Especificação (cont.)

L	:=	Lista_E]	TipoInt:= Integer(vSimbolos.SimboloPorNome(Lista_E*array*).SimboloElemento.Tipo);
			L.local := novo_t; L.Deslocamento := novo_t;
			L.Codigo := gerar(L.local := c(Lista_E*array*);
			L.Codigo := gerar(L.deslocamento || ':=' || Lista_E.local || '*' || Largura(Lista_E.*array*));
			L.CodAsm:=Lista_E.CodAsm || 'MOV DX,' || Lista_Elocal || 'MOV AL,' || IntToStr(TtipoInt) || 'MUL DX' || 'MOV ' || Ldesloca || ',AX';
			se Simb.Tipo = tsMatriz então CodigoAsm := CodigoAsm || 'ADD' || LDesloca || ',' || IntToStr(Lc) //LC negativo
			senao // Matriz Dinâmica
			inicio se (Simb.Escopo.Nivel>0) and (Simb.Escopo.Nivel< vSimbolos.EscopoAtual.Nivel) entao
			CodigoAsm := CodigoAsm || 'PUSH DI' ||
			GerarEloAcesso(vSimbolos.EscopoAtual.Nivel – Simb.Escopo.Nivel) || 'MOV SI,' || Simb.ObtemCodigoAssembly(True,'DI','') || 'POP DI'
			senao CodigoAsm := CodigoAsm || 'MOV SI,' || Simb.ObtemCodigoAssembly(True,'BP','');
			CodigoAsm := CodigoAsm || 'SUB' || Ldesloca || ',[SI+4]'; // O valor está positivo;
			Fim;

	|	#ID	L.local := ID.local; L.deslocamento := '';
Lista_E	::=	ID[Expressao	R_i.matriz:=ID.local; R_i.local:=Expressao.local; R_i.ndim:=1;
		R	Lista_E.matriz:= R_s.matriz; Lista_E.local:= R_s.local; Lista_E.ndim:= R_s.ndim;

Especificação (cont.)

F	::=	'('	Expressao.v := F.v; Expressao.f := F.f
		Expressao	
		')'	F.local := Expressao.local; F.Codigo := Expressao.Codigo; F.CodAsm := Expressao.CodAsm;
	|	'-'	
		Expressao	F.local := novo_t; F.Codigo := Expressao.Codigo || gerar(F.local || ':=' || 'uminus' || E.local);
	|	'NAO'	Expressao.v := F.f; Expressao.f := F.v;
		Expressao	
	|	L	Simb := vSimbolos.ProcurarSimbolo(Evar); se Simb <> nil and (Simb.Tipo=tsMatrizDinamica) then inicio F.Local := NOVO_t; se (Simb.Escopo.Nivel>0) and (Simb.Escopo.Nivel< vSimbolos.EscopoAtual.Nivel) entao F.CodAsm := F.CodAsm || 'PUSH DI' || GerarEloAcesso(vSimbolos.EscopoAtual.Nivel – Simb.Escopo.Nivel) || 'MOV SI,' || Simb.ObtemCodigoAssembly(True,'DI',LDesloca) 'POP DI; senao F.CodAsm := F.CodAsm || 'MOV SI,' || Simb.ObtemCodigoAssembly(True,'BP',LDesloca); F.CodAsm := F.CodAsm || 'MOV SI,[SI]' || 'ADD SI,' || LDesloca || 'MOV' || F.Local || ',[SI]'; Fim; senao se (Simb <> nil) então se L.deslocamento := 0 então F.local := L.local senão F.local := novo_t; gerar(F.local ':=' L.local '[' L.deslocamento ']');

Especificação (cont.)

F	::=	NUM	F.local := NUM; F.Codigo := ''; F.CodAsm := '';
	|	**FDimensoes**	**F.Local := FDimensoes.Local;** **F.Codigo := FDimensoes.Codigo;** **F.CodAsm := FDimensoes.CodAsm;**
	|	**FLimInf**	**F.Local := FLimInf.Local;** **F.Codigo := FLimInf.Codigo;** **F.CodAsm := FLimInf.CodAsm;**
	|	**FLimSup**	**F.Local := FLimSup.Local;** **F.Codigo := FLimSup.Codigo;** **F.CodAsm := FLimSup.CodAsm;**
	|	^	

Especificação (cont.)

R	::=	Expressao	t:='DI';
			m:=ndim+1;
			Simb := vSimbolos.SimboloPorNome(L_E*Array*);
			se Simb.Tipo = tsMatriz entao
			  ValorDimensao := IntToStr(limite(L_E*array*,m));
			senao
			inicio // Matriz Dinâmica
			  se (Simb.Escopo.Nivel>0) and (Simb.Escopo.Nível
			  vSimbolos.EscopoAtual.Nivel) entao
			  R.CodAsm := R.CodAsm ||
			  GerarEloAcesso(vSimbolos.EscopoAtual.Nivel –
			  Simb.Escopo.Nivel) || 'MOV DI,' ||
			  Simb.ObtemCodigoAssembly(True,'DI','');
			  senao
			  R.CodAsm := R.CodAsm || 'MOV DI,' ||
			  Simb.ObtemCodigoAssembly(True,'BP','');
			  R.CodAsm := R.CodAsm ||
			  'ADD DI,6+ || IntToStr((m - 1) * 4) || 'MOV AX,[DI+2]' ||
			  'SUB AX,[DI]' || 'INC AX' || 'MOV DI,AX';
			  ValorDimensao := 'DI';
			fim;
			gerar(t + ':=' + local + '*' + ValorDimensao);
			R.CodAsm := R.CodAsm || Expressao.CodAsm ||
			  'MOV AX,' || ValorDimensao || 'MOV DI,SI' || 'MUL DI' ||
			  'MOV' || t || ',AX' || 'ADD' || t || ',' || Expressao.local;
			R.local:=t; gerar(t || ':=' || t || '+' || Expressao.local);

		R_1	R_s.matriz := R_1.matriz; R_s.local := R_1.local; R_s.ndim := R_1.ndim; R_s.CodAsm := R_1.CodAsm;
	|	^	

Especificação (cont.)

```
M[i,j] := dimensoes(M);
```

FDimensoes	::=	'DIMENSOES'	
		'('	

		#ID	Simb := vSimbolos.ProcurarSimbolo(Nome);
			se nao Simb <> nil então
			  Erro('Identificador não declarado');
			Se Simb.Tipo <> tsMatrizDinamica então
			  Erro('Identificador não é uma matriz dinâmica');

		')'	FDimensoes.Local := novo_t;
			se (Simb.Escopo.Nivel > 0) and (Simb.Escopo.Nivel <
			  vSimbolos.EscopoAtual.Nivel) entao
			  FDimensoes.CodAsm := 'PUSH DI' ||
			  GerarEloAcesso(vSimbolos.EscopoAtual.Nivel –
			  Simb.Escopo.Nivel) || 'MOV SI,' ||
			  Simb.ObtemCodigoAssembly(True, 'DI', '') ||
			  'POP DI'
			senao
			  FDimensoes.CodAsm := ‘MOV SI,' ||
			  Simb.ObtemCodigoAssembly(True, 'BP', '');
			FDimensoes.CodAsm := FDimensoes.CodAsm ||
			  'MOV' || Local || ',[SI+2]';

Especificação (cont.)

```
i := LimInf(M, 1);
```

FLimInf	::=	'LIMINF'	
		'('	

		#ID	Simb := vSimbolos.ProcurarSimbolo(Nome);
			se nao Simb <> nil entao
			  Erro'Identificador não declarado');
			se Simb.Tipo <> tsMatrizDinamica entao
			  Erro('Identificador não é uma matriz dinâmica');

		','	

		Expressao	se Expressao.Tipo <> tsInteiro então
			  Erro('Tipo incompatível');
			Local := novo_t;
			FLimInf.CodAsm := FLimInf.CodAsm ||
			  Expressao.CodAsm || Expressao.PreLocal ||
			  'MOV AX,' || Expressao.Local ||
			  'DEC AX' || 'MOV DI,2' || 'MUL DI' || 'MUL DI';
			se (Simb.Escopo.Nivel > 0) and (Simb.Escopo.Nivel <
			  vSimbolos.EscopoAtual.Nivel) entao
			  FLimInf.CodAsm := FLimInf.CodAsm ||
			  GerarEloAcesso(vSimbolos.EscopoAtual.Nivel –
			  Simb.Escopo.Nivel) || 'MOV SI,' ||
			  Simb.ObtemCodigoAssembly(True, 'DI', '');
			senao
			  FLimInf.CodAsm := FLimInf.CodAsm || 'MOV SI,' ||
			  Simb.ObtemCodigoAssembly(True, 'BP', '');
			
			FLimInf.CodAsm := FLimInf.CodAsm +
			  'ADD SI,6'#10 || 'ADD SI,AX'#10 || 'MOV' ||
			  Local || ',[SI]';

		')'	

Especificação (cont.)

```
j := LimSup(M, 2)
```

FLimSup	::=	'LIMSUP'	
		'('	

		#ID	Simb := vSimbolos.ProcurarSimbolo(Nome);
			se nao Simb <> nil entao
			  Erro'Identificador não declarado');
			se Simb.Tipo <> tsMatrizDinamica entao
			  Erro('Identificador não é uma matriz dinâmica');

		','	

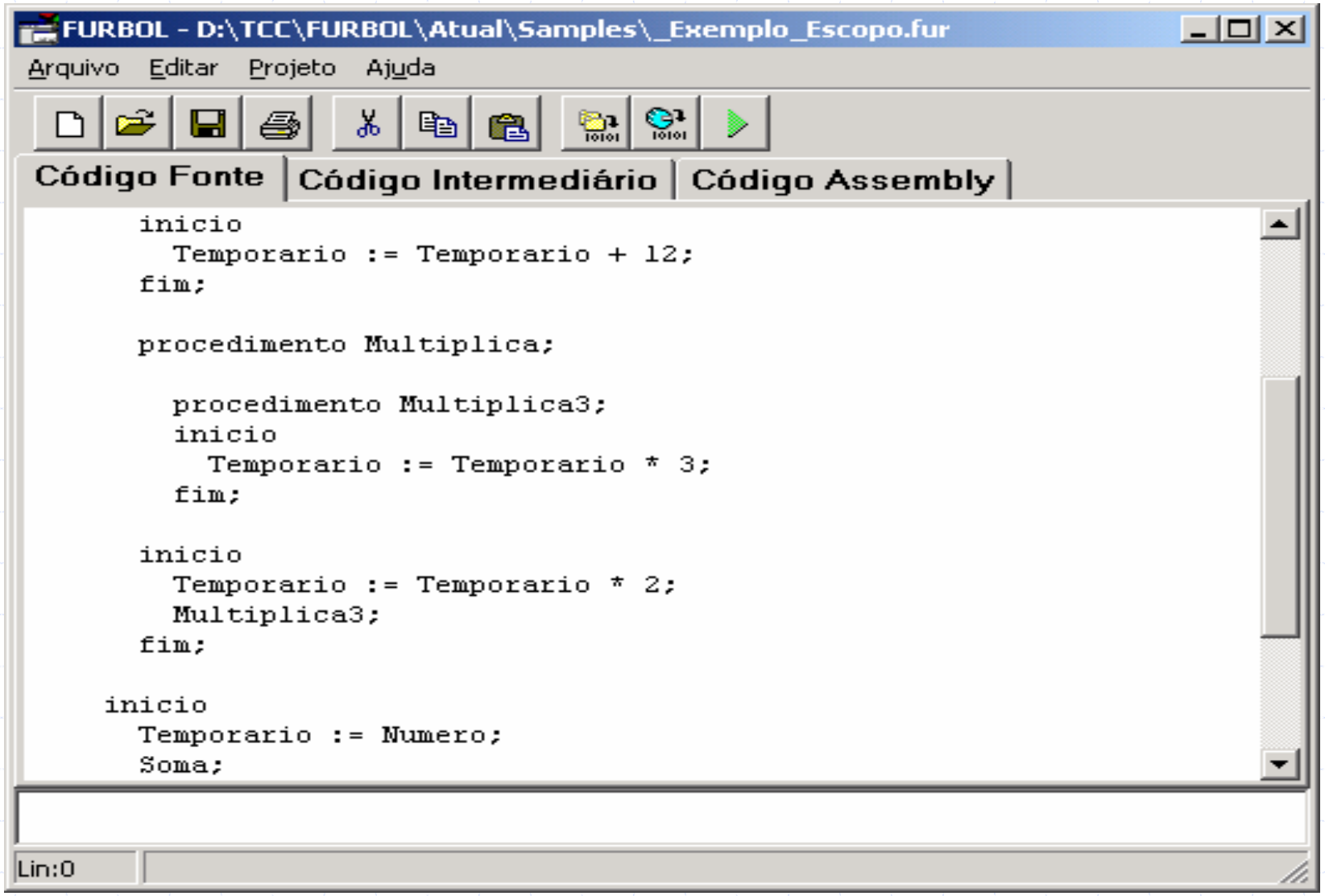
		Expressao	se Expressao.Tipo <> tsInteiro então
			  Erro('Tipo incompatível');
			Local := novo_t;
			  FLimSup.CodAsm := FLimSup.CodAsm ||
			  Expressao.CodAsm || Expressao.PreLocal ||
			  'MOV AX,' || Expressao.Local ||
			  'DEC AX' || 'MOV DI,2' || 'MUL DI' || 'MUL DI';
			se (Simb.Escopo.Nivel > 0) and (Simb.Escopo.Nivel <
			  vSimbolos.EscopoAtual.Nivel) entao
			  FLimSup.CodAsm := FLimSup.CodAsm ||
			  GerarEloAcesso(vSimbolos.EscopoAtual.Nivel –
			  Simb.Escopo.Nivel) || 'MOV SI,' ||
			  Simb.ObtemCodigoAssembly(True, 'DI', '');
			senao
			  FLimSup.CodAsm := FLimSup.CodAsm || 'MOV SI,' ||
			  Simb.ObtemCodigoAssembly(True, 'BP', '');
			FLimSup.CodAsm := FLimSup.CodAsm +
			  'ADD SI,6'#10 || 'ADD SI,AX'#10 || 'MOV' ||
			  Local || ',[SI+2]';

		')'	

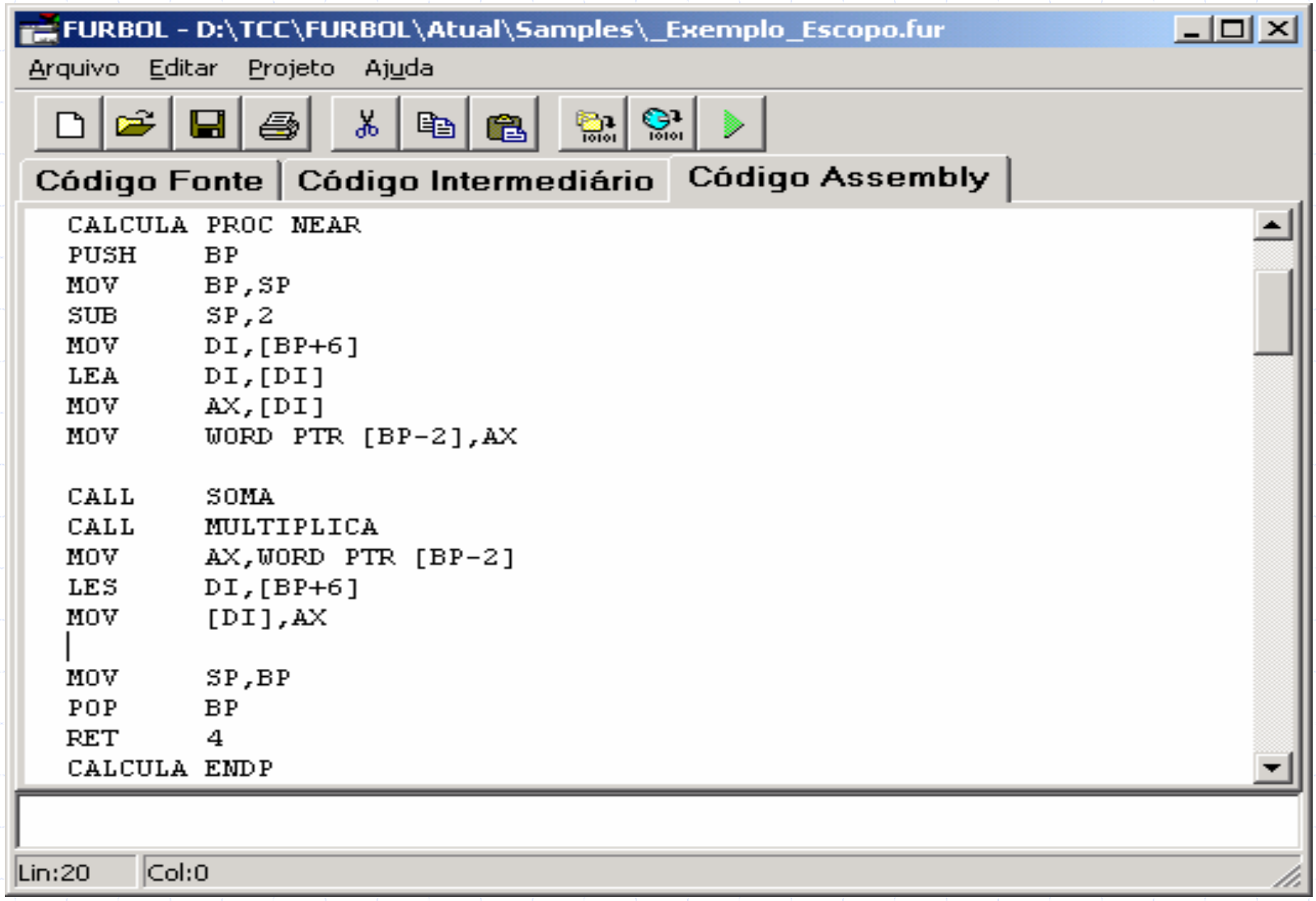
Operacionalidade Protótipo

- Principais características do ambiente foram mantidas (funções da interface)
- A formatação do código *Assembly* gerado foi melhorada
- Principais alterações deste trabalho
  - Implementação de mapeamento finito dinâmico
  - Acesso à variáveis não-locais

Interface do Protótipo



Interface do Protótipo



Exemplo de Uso de *Array* Dinâmico

```
programa Exemplo;
 var
   m: Matriz: Inteiro;
   i, j: Inteiro;
 inicio
   Redimensiona(M, 1..10, 1..10);
   i := LimInf(M, 1);
   enquanto i < LimSup(M, 1) faca
   início
     j := LimInf(M, 2);
     enquanto j < LimSup(M, 2) faca
     inicio
       M[i,j] := i * j;
       Inc(j);
     fim;
     M[i,j] := dimensoes(M);
     Inc(i);
   fim;
   M[i,j] := dimensoes(M);
 fim.
```

Exemplo de Acesso à Variável Não-Local

```
procedimento Calcula(ref Numero: Inteiro);
var Temporario: Inteiro;

  procedimento Soma;
  inicio
    Temporario := Temporario + 12;
  fim;

  procedimento Multiplica;
    procedimento Multiplica3;
    inicio
      Temporario := Temporario * 3;
    fim;
  inicio
    Temporario := Temporario * 2;
    Multiplica3;
  fim;

inicio
  Temporario := Numero;
  Soma; Multiplica;
  Numero := Temporario;
fim;
```

Conclusões

- Teoria ajudou o entendimento do trabalho
- Principais características
  - Elementos e funções de *arrays* dinâmicos podem ser usados como operandos em expressões
  - Expressões podem ser usadas em subscritos
- Dificuldades
  - Adaptação da especificação ao novo escopo implementado
- Limitações
  - Atribuições entre *arrays* (dinâmicos e semi-estáticos)
  - Verificação de limites para subscritos

Extensões

- Unidades do tipo função
- Comandos de entrada e saída de dados
- Comando de repetição *Para*
- Comando de seleção *Caso*
- Geração de código no formato *EXE*
- Otimização do código gerado