IMPLEMENTAÇÃO DE MAPEAMENTO FINITO (*ARRAY'S*) NO AMBIENTE FURBOL

Acadêmico: Anderson Adriano

Orientador: José Roque Voltolini da Silva

Roteiro da Apresentação

- motivação;
- introdução;
- objetivos;
- compiladores;
- análise léxica;
- análise sintática;
- análise semântica;
- mapeamento finito (*array's)*

- *array's* e índices;
- categorias de *array's*
- implementação do tipo *array*;
- organização de *array's*;
- fórmula para *array's* com uma ou mais dimensões;
- exemplo de declaração de *array's* no ambiente FURBOL;
- tabelas de símbolos para declarações *array*;

- exemplo de declaração e uso de *array's* no ambiente FURBOL;
- tradução em código *Assembler;*
- conclusão;
- extensões.

Motivação

- A escolha do assunto do trabalho foi motivada por ser continuação de um projeto iniciado em 1987, o qual vem sendo aprimorado através de trabalhos de conclusão de curso.

Introdução

- A concepção inicial do ambiente FURBOL teve inicio em 1987 através da experiência relatada no artigo Silva (1987), apresentado no I Simpósio de Engenharia de Software;
- em 1993, houve uma continuidade através do Programa de Iniciação a Pesquisa (PIPe);
- Após, o ambiente FURBOL vem se extendo através de trabalhos de conclusão de curso.

Objetivos

- Ampliação do ambiente FURBOL através da inclusão de mapeamento finito (*array's);*
- Geração de código executável.

Compiladores

- Segundo Aho (1995), posto de forma simples, um compilador é um programa que lê um programa em uma linguagem, e o traduz num programa equivalente numa outra linguagem chamada de *linguagem-alvo*.

Análise Léxica

- Constitui a atividade que faz a interface entre o texto-fonte e os programas encarregados de sua análise e tradução.

Análise Sintática

- O segundo grande bloco componente dos compiladores, e que se pode caracterizar como o mais importante, na maioria dos compiladores, por sua característica de controlador das atividades do compilador.

Definições Dirigidas por sintaxe

- Uma definição dirigida pela sintaxe é aquela na qual cada símbolo gramatical possui um conjunto de atributos, particionados em dois subconjuntos, chamados de atributos sintetizados e atributos herdados daquele símbolo gramatical.

Gramática de Atributos

- Uma gramática de atributos é uma definição dirigida pela sintaxe, na qual as funções nas regras semânticas não têm efeitos colaterais, ou seja, estas funções não alteram seus parâmetros ou variável não local (Knuth, 1968).

Atributos Sintetizados

- Um atributo é dito sintetizado se seu valor em um nó da árvore gramatical é determinado a partir dos valores dos atributos dos filhos daquele nó.

Atributos herdados

- Um atributo herdado, segundo Aho (1995), é aquele cujo valor a um nó de uma árvore gramatical é definido em termos do pai e/ou irmãos daquele nó.

Análise Semântica

- A geração do código vem acompanhada das atividades de análise semântica, responsáveis pela captação do sentido do texto-fonte.

Mapeamento Finito (*Array's*)

- Um mapeamento finito é conhecido como correspondência. Nas linguagens de programação normalmente conhecido como definições de matrizes (quando de uma dimensão, normalmente chamada de vetor).

Array's e índices

- Elementos de um *array* são referenciados por meio de um mecanismo sintático de dois níveis, cuja a primeira parte é o nome do agregado e a segunda é um seletor que consiste em um ou mais itens conhecidos como subscritos ou índices.

nome_do_array(lista_de_valores_indice) → elemento

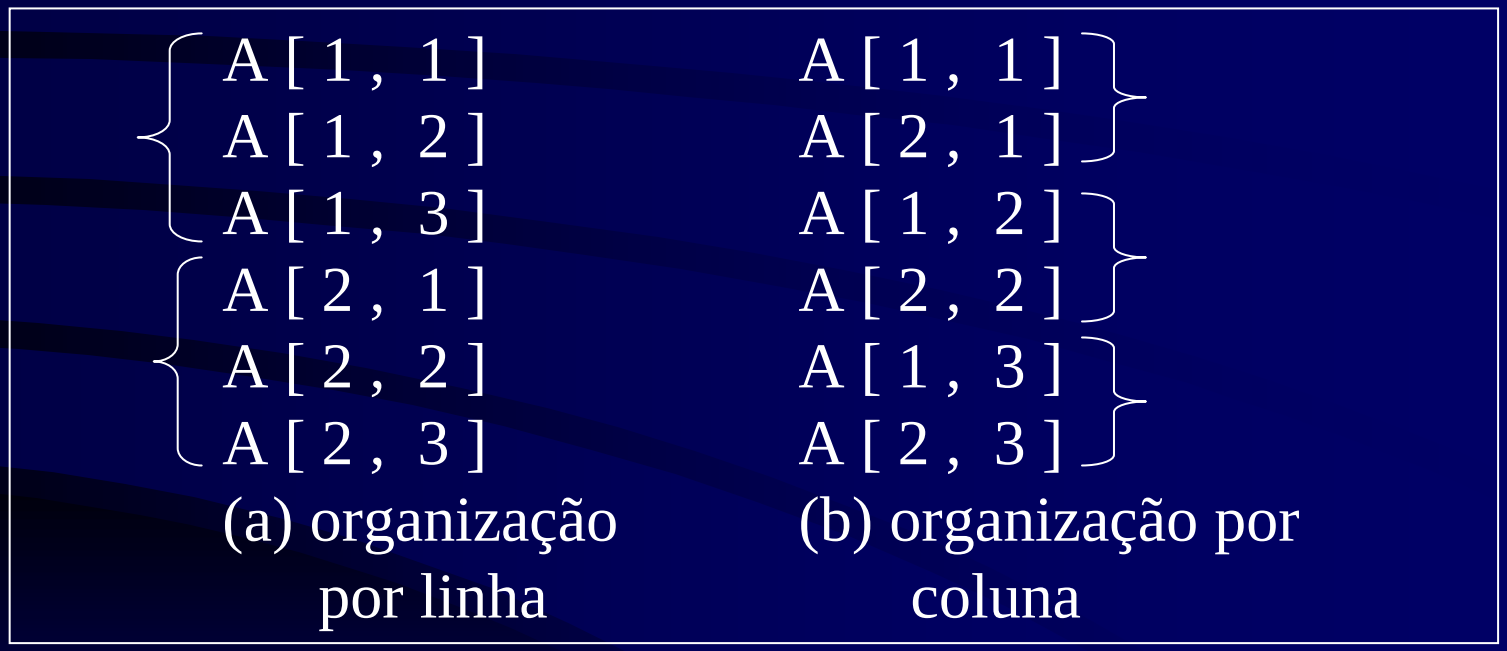
Categorias de *array's*

- *array* estático;
- *array* stack-dinâmico fixo (semi-estático);
- *array* stack-dinâmicos (semi-dinâmico);
- *array* heap-dinâmico (dinâmico).

Implementação do tipo *array*

- Os elementos de um *array* podem receber acesso mais rapidamente se os elementos forem armazenados num bloco de localizações cosectutivas.

Organização de *arrays's*



Fórmula para *array's* com uma ou mais dimensões

$$((\ldots((i_1\, n_2 + i_2)\, n_3 + i_3)\ldots)\, n_K + i_K)\, X\ w$$
$$+\ base\ -\ ((\ldots((linf_1\, n_2 + linf_2)\, n_3 + linf_3)\ldots)\, n_K + linf_K)\, X\, w$$

```
programa teste;
Var
   M1:matriz[1..2,1..3]:inteiro;

Inicio
   M1[2,3]:=7;
fim.
```

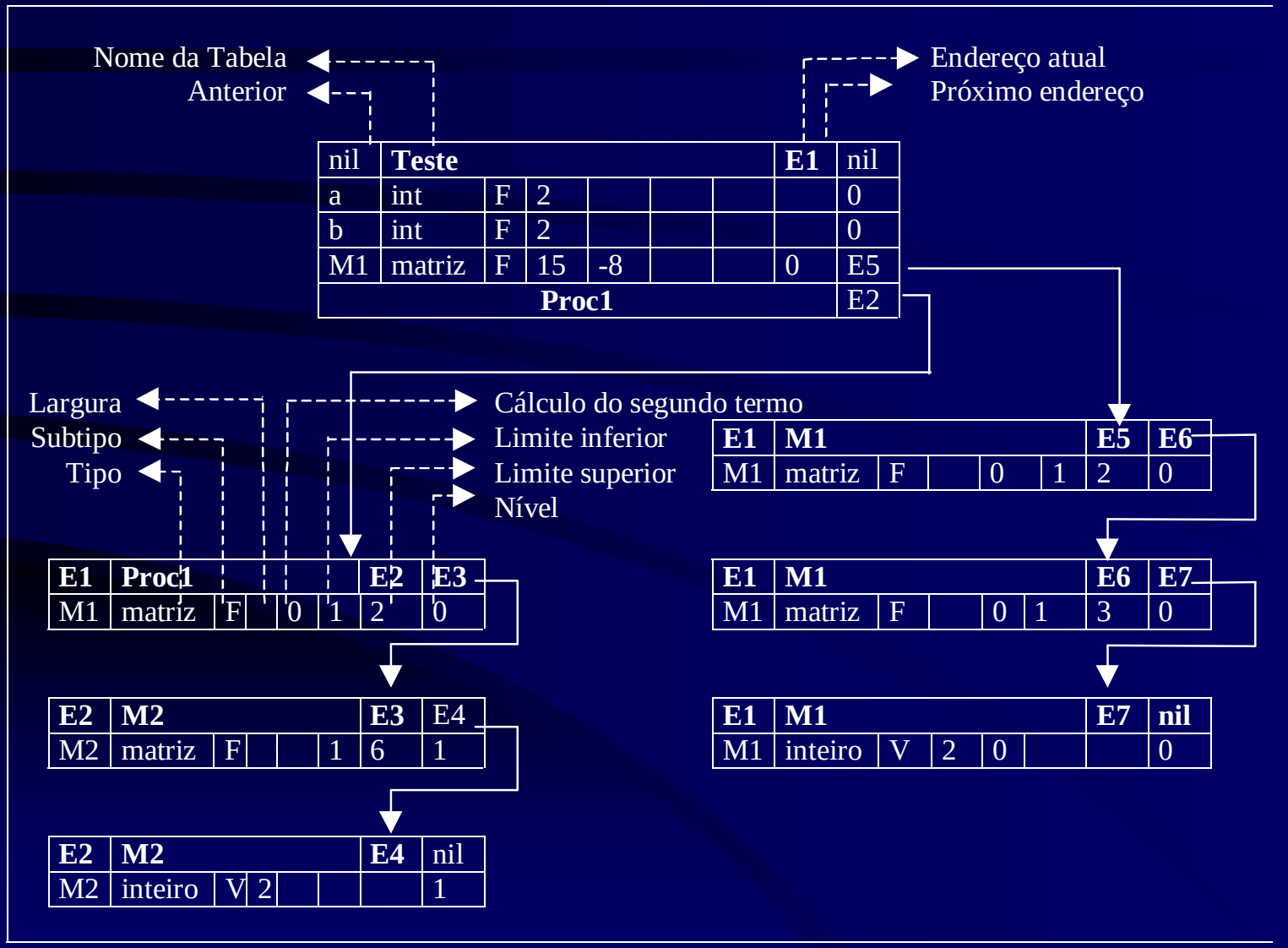
$$(i_1\, n_2 + i_2)\ X\ w + base - (linf_1\, n_2 + linf_2)\, X\, w$$

$$(2 * 3 + 3) * 2 + 0 - (1 * 3 + 1) * 2 = 10$$

Exemplo de declaração de *array's* no ambiente FURBOL

```
programa teste;
Var
   a,b:inteiro;
   M1:matriz[1..2,1..3]:inteiro;
  procedimento proc1;
     var
     M2:matriz[1..6]:inteiro;
    Inicio
     fim;
Inicio
fim.
```

Tabelas de símbolos para declarações *array*



Definição de Programas e Blocos

Programa	::=	‘PROGRAMA’,	GeraAsm('CODIGO SEGMENT'); GeraAsm('ASSUME CS:CODIGO, DS:CODIGO'); GeraAsm('ORG 100H');
		#ID,	ID.nome:=#ID; GeraAsm('ENTRADA: JMP ' || #ID.nome); nivel:=0;
		‘;’,	T:=CriarTabela(nil); Empilhar(t,ptrtab);Empilhar(0,Deslocamento);
		EstruturaDados,	nível:=nivel+1;
		EstruturaSubRotinas,	nivel:=nivel-1;
		CComposto	GeraAsm(EstruturaSubRotinas.codAsm || ID.nome ||' PROC NEAR' || 'PUSH BP' || 'MOV BP,SP' || CComposto.CodAsm || 'POP BP' || 'int 20h' || ID.nome || ENDP);

		‘.’	DeclaraDados; // Variáveis globais em Assembly
			GerarASm(CODIGO ENDS);
			GerarAsm(END ENTRADA);
			RegistrarLarg(Topo(PtrTab),Topo(Deslocamento));
			Desempilha(PtrTab);Desempilha(Deslocamento);

Definição de Programas e Blocos (continuação)

Bloco	::=	EstruturaDados,	
		EstruturaSubRotinas,	
		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’;	

Definição das Estruturas de Dados

EstruturaDados	::=	‘VAR’,	
		#ID,	Se não (encontra_var(ID)) então Instalar(TopoPtrTab),ID.Nome);
		ListaID, (Matriz|^)	AtualizarTipo(Topo(PtrTab),NroVar,T.tipo,Topo(Deslocamento),TipoDesloca,LimInf,LimSup,SubTipo); Topo(Deslocamento):=Topo(Deslocamento)+T.Largura;
		‘;’,	
		Declarações	
		| ^;	
Declarações	::=	#ID,	Se não (encontra_var(ID)) então Instalar(TopoPtrTab),ID.Nome,ID.tipo,param);
		ListaID, (Matriz|^)	AtualizarTipo(Topo(PtrTab),NroVar,T.tipo,Topo(Deslocamento),TipoDesloca,LimInf,LimSup,SubTipo); Topo(Deslocamento):=Topo(Deslocamento)+T.Largura;
		‘;’,	
		Declarações	
		|^;	
ListaID	::=	‘,’,	
		ID,	Se não (encontra_var(ID)) então Instalar(TopoPtrTab),ID.Nome,ID.Tipo,param);
		ListaID, (Matriz|^)	AtualizarTipo(Topo(PtrTab),NroVar,T.tipo,Topo(Deslocamento),TipoDesloca,LimInf,LimSup,SubTipo); Topo(Deslocamento):=Topo(Deslocamento)+T.Largura;
		| ‘:’,	
		Tipo;	ListaId.Tipo:=Tipo;

Definição das Estruturas de Dados (continuação)

Tipo	::=	‘INTEIRO’	T.Tipo:=2; T.Largura:=2;
		|	
		‘LOGICO’;	T.Tipo:=1;T.Largura:=1;
		|	
		‘MATRIZ’’	T.Tipo:=4;
Matriz	::=	LimitesM, ‘:’, Tipo	AtualizarTipo(Topo(PtrTab),NroVar,T.tipo,Topo(Deslocamento),TipoDesloca,LimInf,LimSup,SubTipo); Topo(Deslocamento):=Topo(Deslocamento)+T.Largura;
		| ^	
LimitesM	::=	‘[‘, Dimensao, ‘]’	
Dimensao	::=	#NUM, ‘..’, #NUM,	T:=CriarTabela(nil); Empilhar(t,ptrtab);Empilhar(0,Deslocamento); Instalar(Mid,Mtipo,Mparam,Lsup,Linf);
		MaisDimensao	

Definição das Estruturas de Dados (continuação)

MaisDimensao	::=	‘,’, Dimensao	
	|		
		^	

Definição da estrutura de Atribuição

Atribuição	::=	‘:=’,	Se Atribuição.deslocamento <> ‘’ então RX:=novo_t; IndCodAsm:='MOV RX,’ || Ldesloca || 'PUSH 'RX; PreIdAt:='POP DI';
		Expressão;	E.local := Expressão.local; se Expressao.tipo:<>Atribuicao.tipo entao erro; se Expressão.deslocamento :<> ‘ ’ então Atribuição.código:=gerar(idAT’[‘L.deslocamento ‘]:=’E.local); senão Atribuição.código:=gerar(IdAT ':=' E.Local); Atribuição.Asm:= IndCodAsm Expressao.CodAsm || AtPrelocal 'MOV RX,'aTlocal || PreIdAT || ‘MOV ‘IdAT','RX;

Definição da Estrutura de Controle de Expressões

F	::=	‘(‘	Expressão.v := F.v Expressão.f := F.f
		Expressão,	
		‘)’	F.local := Expressão.local; F.código := Expressão.código F.códAsm := Expressão.códAsm
		|	
		‘-‘,	
		Expressão	F.local := novo_t; F.código := Expressão.código || gerar(F.local ‘:=’ ‘ uminus ’ E.local)
		|	
		‘NAO’	Expressão.v := F.f; Expressão.f:= F.v;
		Expressão	
		|	
		L	se encontra_var(ID) então se L.deslocamento := 0 então F.local := L.local senão F.local := novo_t; gerar(F.local ’:=’ L.local ‘[‘ L.deslocamento ‘]’);
		|	
		NUM	F.local := NUM; F.código := ‘’; F.códAsm := ‘’;
		| ^;	

Definição da Estrutura de Controle de Expressões (continuação)

L	:=	Lista_E]	L.local := novo_t; L.deslocamento := novo_t; L.código := gerar(L.local := c(Lista_Earray); L.código := gerar(L.deslocamento ':=' Lista_E.local '*' Largura(Lista_E.array)); L.CódAsm:=Lista_E.Códsm+ 'MOV DX,'+Lista_Elocal || 'MOV AL,'+IntToStr(VerificaTipoMatriz(Lista_Earray)) || 'MUL DX' || 'MOV '+LDesloca+',AX' || 'ADD '+LDesloca+','+IntToStr(Lc) || 'ADD '+LDesloca+',2';
		|	
		#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;

Definição da Estrutura de Controle de Expressões (continuação)

R	::=	Expressao	t:=novo_t;
			m:=Ri..ndim+1;
			gerar(t+':='+Lista_E.local*
			limite(Lista_E.matriz,m);
			R_1i.matriz:=Ri.matriz;
			R_1i.local:=t;
			R_1i.ndim:=m;
			R_1i.CódAsm:=Expressao.CódAsm+
			'MOV CX,'+local ||
			'MOV AL,'+IntToStr(limite(L_Earray,m)) ||
			'MUL CX' ||
			'MOV '+t+',AX' ||
			'ADD '+t+','+Elocal;

		R_1	R_s.matriz:= R_{1s}.matriz; R_s.local:= R_{1s}.local; R_s.ndim:= R_{1s}.ndim; R_s.CódAsm:= R_{1s}.CódAsm;
		|	
		^	R_s.matriz:= R_i.matriz; R_s.local:= R_i.local; R_s.ndim:= R_i.ndim; R_s.CódAsm:= R_i.CódAsm;

Tradução em código *Assembler*

```
(1)    CODIGO     SEGMENT
(2)    ASSUME     CS:CODIGO, DS:CODIGO
(3)    ORG        100H
(4)    ENTRADA:   JMP    XPTO6
(5)    K      DW ?
(6)    I      DW ?
(7)    J      DW ?
(8)    B      DW 24 DUP (0)
(9)    A      DW 2 DUP (0)
(10)
(11)   XPTO8  PROC NEAR
(12)   PUSH  BP
(13)   MOV   BP,SP
(14)
(15)   MOV   AX,2
(16)   MOV   K,AX       ;K=2
(17)
(18)   MOV   AX,1
(19)   MOV   I,AX       ;I=1;
(20)
(21)   MOV   AX,1
(22)   MOV   J,AX       ;J=1
(23)                    ;linha 20 à 26 – cálculo do índice do array A
(20)   MOV   DI,K
(21)   MOV   AX,2       ; linha 20 à 23 – primeiro termo da fórmula
(22)   MUL   DI         2 * 2                    ( i1 ) X w
(23)   MOV   DI,AX      + 0 - (1) * 2  = 2       + base  -  ( linf1)  X w
(24)   ADD   DI,-2      ; -2 segundo termo da fórmula
(25)   MOV   BX,DI
(26)   PUSH  BX         ; armazena o cálculo do índice do array A na pilha
(27)
(28)   MOV   AX,1
(29)   POP   DI         ; retira da pilha o valor já calculado do índice
(30)   MOV   A[DI],AX   ; move 1 p/ A[2]
(31)
```

A

```
programa xpto6;
Var
  A:matriz[1..2]:inteiro;

B:matriz[1..2,1..3,1..5]:inteir
o;
  i,j,k:inteiro;

Inicio
 k := 2;
 i := 1;
 j := 1;
 A[k] := 1;
 B[A[k], j*k, 4] := 10;
fim.
```

$$((\ldots((i_1\, n_2 + i_2)\, n_3 + i_3)\ldots)\, n_K + i_K)\, X\, w$$

$$+\ base\ -\ ((\ldots((linf_1\, n_2 + linf_2)\, n_3 + linf_3)\ldots)\, n_K + linf_K)\, X\, w$$

Exemplo de declaração e uso de *array's* no ambiente FURBOL

```
programa xpto6;
Var
  A:matriz[1..2]:inteiro;
  B:matriz[1..2,1..3,2..5]:inteiro;
  i,j,k:inteiro;

Inicio
 k:=2;
 i:=1;
 j:=1;
 A[k]:=1;
 B[A[k],j*k,4]:=10;
fim.
```

Tradução em código *Assembler* (continuação)

```
(32)  MOV  DI,K        ; K=2     linha 32 à 57 – cálculo do índice do array B
(33)  MOV  AX,2
(34)  MUL  DI
(35)  MOV  DI,AX       ; 2 * 2
(36)  ADD  DI,-2       ; + 0 - (1) * 2  = 2
(37)  MOV  CX,A[DI]    ; CX=1
(38)  MOV  SI,CX
(39)  MOV  DI,K
(40)  MOV  AX,J        ; J * K
(41)  MUL  DI
(42)  MOV  BX,AX       ; BX=2
(43)  MOV  DI,SI       ; DI=1

(44)  MOV  AX,3
(45)  MUL  DI
(46)  MOV  DI,AX
(47)  ADD  DI,BX
(48)  MOV  SI,DI       ;SI=5
(49)  MOV  DI,SI
(50)  MOV  AX,5
(51)  MUL  DI
(52)  MOV  DI,AX
(53)  ADD  DI,4
(54)  MOV  AX,2
(55)  MUL  DI
(56)  MOV  DI,AX       ; DI = 58
(57)  ADD  DI,-44
(58)  MOV  CX,DI       ; CX = 14
(59)  PUSH CX          ; empilha CX (cálculo do índice do array B)
(60)  MOV  AX,10
(61)  POP  DI          ; desempilha para DI o cálculo do índice do array B
(62)  MOV  B[DI],AX    ; B[14] = 10
(63)  POP  BP
(64)  Int  20h
(65)  XPTO6  ENDP
(66)  CODIGO  ENDS
(67)  END  ENTRADA
```

A

B

$(i_1)\ X\ w$

$+ base\ -\ (linf_1)\ X\ w$

$((\ i_1\ n_2 +\ i_2\ )\ n_3 +\ i_3\ )\ X\ w$

$+ base\ -\ ((\ linf_1\ n_2 +\ linf_2\ )\ n_3 + linf_3\ )\ X\ w$

((1 * 3 + 2) * 5 + 4) * 2 +

0 - ((1 * 3 + 1) * 5 + 2) * 2 = 14

```
programa xpto6;
Var
  A:matriz[1..2]:inteiro;

B:matriz[1..2,1..3,1..5]:inteir
o;
  i,j,k:inteiro;

Inicio
 k:=2;
 i:=1;
 j:=1;
 A[k]:=1;
 B[A[k],j*k,4]:=10;
fim.
```

$((\ ...\ ((\ i_1\ n_2 +\ i_2\ )\ n_3 +\ i_3\ )\ ...\ )\ n_K\ + i_K\ )\ X\ w$

$+ base\ -\ ((\ ...\ ((\ linf_1\ n_2 +\ linf_2\ )\ n_3 + linf_3\ )\ ...\ )\ n_K + linf_K\ )\ X\ w$

Conclusão

- os objetivos do trabalho foram alcançados, os quais eram a definição e implementação de mapeamento finito (*array's*);
- estas novas inclusões ampliam a capacidade do compilador FURBOL, tornando-o mais abrangente;

Extensões

- a estrutura do tipo *registro* não foi implementado neste trabalho, bem como comandos de entrada e saída de dados. Sendo estas implementações, sugestões para trabalhos futuros;
- Outras sugestões para trabalhos futuros são a implementação de unidades do tipo função (*function* no Pascal) e a ampliação do uso de *array's* para outros tipos além do inteiro;

- Também como sugestão, fazer a otimização código *assembly* gerado pelo protótipo e um gerador de código executável (montador) que tenha como saída um arquivo no formato *.EXE*.