Composite data types: vectors and matrices

Lecture 6

Formal Languages and Compilers 2011

Nataliia Bielova

Definition

- Data: “container” for values (var or const)
- Value: something that is put in the data (everything that is representable with a sequence bits)
- Data type (DT): class for data and operations to manipulate it

Data

- Categories:
  - Basic data types: integers, floats, characters, enumerable types,…
  - Structured data (data structures): matrices, records, lists,…
- Specification:

  attributes : "technical" aspects for managing data

  values : what you can put inside the data

  operations : what you can do with that data
- Implementation: how the specification is realized in practice

Basic data type: integer

- Specification:
  - attributes : how it is represented in the internal memory
  - values : the maximum and minimum are defined: [MinInt], [MaxInt]
  - operations : sum, multiplication, subtraction, division,…

- Implementation:
  - attributes : decide at compile-time or at run-time
  - values : nothing to declare
  - operations : HW operations: `ADD, MUL, …`
    procedure: `Sum(x,y) = x + y`
    …

Data structure: array

- Specification:

  attributes

  - number of the components
  - type of components
  - a way to access them etc.

  values : decided by the attributes

  operations

  - modify the structure (insert, delete, …)
  - operations over one component
  - operations over the entire structure (comparison, copy)

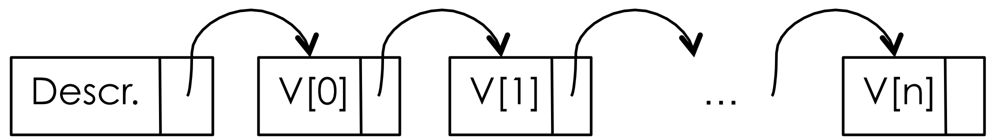
Data structure: array

- Implementation:

  attributes : in the *descriptor*

  values : like before

  operations : access to the elements:



☺ $\Lambda \| V[k] \| = B + O(k)$

☹ Ins.& Del.

☹ $\Lambda \| V[k] \|$ = scanning the whole list

☺ Ins.& Del.

Array in crème CAraMeL

- Data structure
- Homogenous (consists of elements of one type)
- Fixed length → represented by a sequence

Descr.	V[0]	V[1]	…	V[n]

linear array : vector

multidimensional array : matrix (remembered line by line)

Vector in crème CAraMeL

Specification

- attributes
  - number of elements
  - type (dim.) of elements
  - component name = index

Implementation

- attributes
  - `var V:array [LB .. UB] of` *type*
  - *type* -> `M`(ultiplier)
  - `O(k) = M x k`

Vector in crème CAraMeL

Specification

- attributes
  - number of elements
  - type (dim.) of elements
  - component name = index
- values: v. number and type

Implementation

- attributes
  - `var V:array [LB .. UB] of` *type*
  - *type* -> `M`(ultiplier)
  - `O(k) = M x k`
- values: `UB-LB+1` elem. of type *type*

Vector in crème CAraMeL

Specification

- attributes
  - number of elements
  - type (dim.) of elements
  - component name = index
- values: v. number and type
- operations:
  - access to the elements
  - creation/elimination of the vectors

Implementation

- attributes
  - `var V:array [LB .. UB] of` *type*
  - *type* -> `M`(ultiplier)
  - `O(k) = M x k`
- values: `UB-LB+1` elem. of type *type*
- operations:

  $\Lambda\|\mathtt{V[k]}\| = \alpha + (\mathtt{k} - \mathtt{LB}) \times \mathtt{M}$
  - declaration

Vectors: implementation

- Address of the k-th element:

$$\Lambda\|V[k]\| = \alpha + (k - LB) \times M = (\alpha - LB \times M) + k \times M = VO + k \times M$$

$$VO = \alpha - LB \times M = \Lambda\|V[0]\|$$

Descriptor:

VO
LB
UB
M

Representation in the memory:

V[0]	VO
...	
V[LB]	α
V[LB+1]	
...	
V[UB]	

Vectors: implementation

- Address of the k-th element:

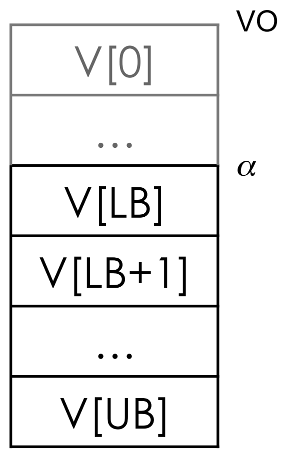
$$\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) = (\alpha - \texttt{LB}) + \texttt{k} = \texttt{VO} + \texttt{k}$$

$$\texttt{VO} = \alpha - \texttt{LB} = \Lambda\|\texttt{V[0]}\|$$

Descriptor:

VO
LB
UB

Representation in the memory:

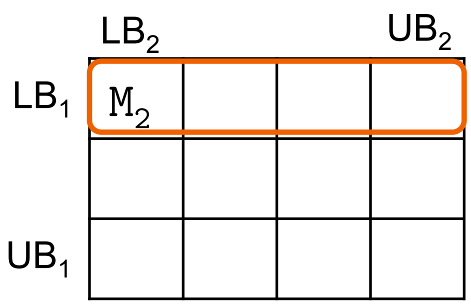


Simplification: $\texttt{M} = 1$

Bidimensional matrices

var V : array[LB_1..UB_1, LB_2..UB_2] of *type*

- Dimension of an element: M_2
- Dimension of a row: $M_1 = (UB_2 - LB_2 + 1) \times M_2$

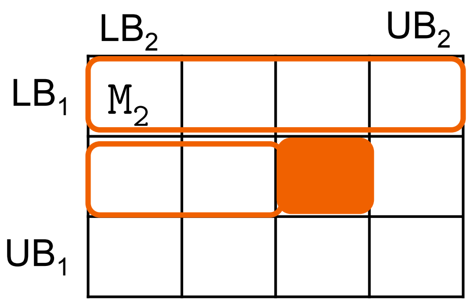


Bidimensional matrices

`var V : array[LB₁..UB₁, LB₂..UB₂] of type`

$var\ V : array[LB_1..UB_1, LB_2..UB_2]\ of\ \mathit{type}$

- Dimension of an element: M_2
- Dimension of a row: $M_1 = (UB_2 - LB_2 + 1) \times M_2$



i=2, j=3

- Virtual Origin: $VO = \alpha - LB_1 \times M_1 - LB_2 \times M_2$

$$\Lambda \| V[i, j] \| = VO + i \times M_1 + j \times M_2$$

Multidimensional matrices

$$\texttt{var V : array[LB}_1\texttt{..UB}_1\texttt{, ..., LB}_n\texttt{..UB}_n\texttt{] of } \textit{type}$$

- Multipliers:

$$M_n = M$$

$$M_i = (UB_{i+1} - LB_{i+1} + 1) \times M_{i+1} \quad i \in [1, n-1]$$

$$VO = \alpha - \sum_{i=1}^{n} LB_i \times M_i$$

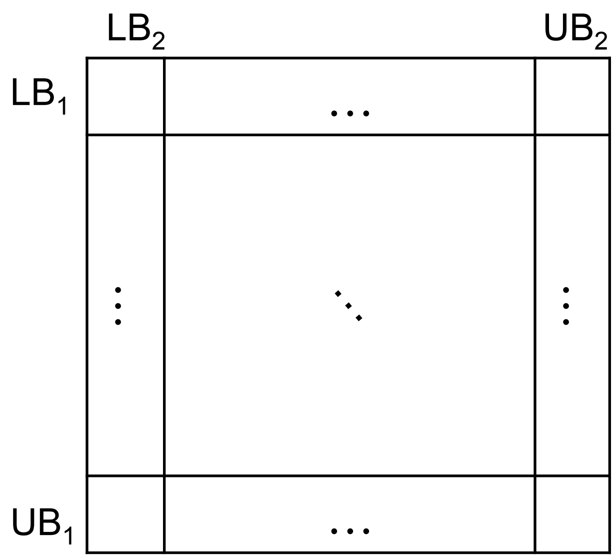
$$\Lambda \| V[k_1, \ldots, k_n] \| = VO + \sum_{i=1}^{n} k_i \times M_i$$

$$\texttt{array[LB}_1\texttt{..UB}_1\texttt{, LB}_n\texttt{..UB}_n\texttt{] of } \textit{type}$$

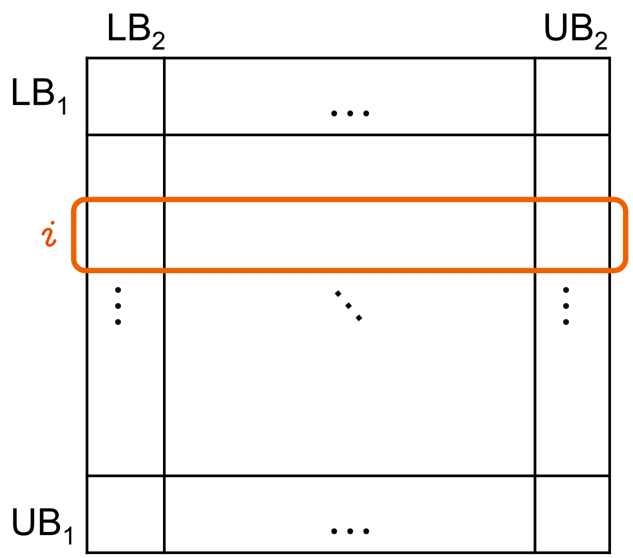
$$\approx$$

$$\texttt{array[LB}_1\texttt{..UB}_1\texttt{] of (array[LB}_2\texttt{..UB}_2\texttt{, LB}_n\texttt{..UB}_n\texttt{] of } \textit{type}\texttt{)}$$

Slices of array



Slices of array



$$M = M_2$$

$$VO_I = VO_V + i \times (UB_2 - LB_2 + 1) \times M_2 =$$

$$VO_V + i \times M_1$$

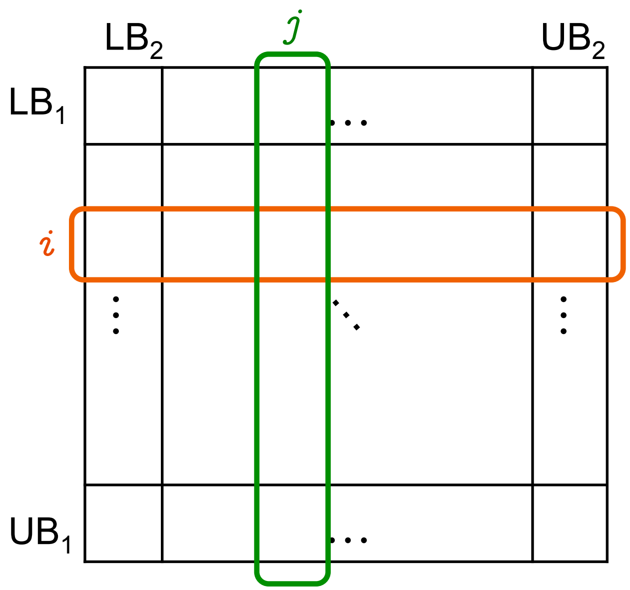
$$LB = LB_2$$

$$UB = UB_2$$

$$\Lambda \| I[k] \| = VO_I + k \times M$$

$$I = V[i][*] = \{V[i][LB_2], V[i][LB_2 + 1], ..., V[i][UB_2]\}$$

Slices of array



$$M = (UB_2 - LB_2 + 1) \times M_2 = M_1$$

$$VO_J = VO_V + j \times M_2$$

$$LB = LB_1$$

$$UB = UB_1$$

$$\Lambda \| J[k] \| = VO_J + k \times M$$

$$I = V[i][*] = \{V[i][LB_2], V[i][LB_2 + 1], \ldots, V[i][UB_2]\}$$

$$J = V[*][j] = \{V[LB_1][j], V[LB_1 + 1][j], \ldots, V[UB_1][j]\}$$

Implementation of array in crème CAraMeL

Syntax:

- parser.mly: new token ARRAY, OF, LBRACKET, RBRACKET, DOTS
- lexer.mll: strings corresponding to new tokens
- syntaxtree.ml: constructors
  - Vector of bType * int * int for declaration

  ```
  var v:array [0..6] of int
  ```

  - LVec of ide * aexp for the left side of the assignment

  ```
  v[0]:=5;
  ```

  - Vec of ide * aexp for expressions

  ```
  x:= v[2];
  ```

- parser.mly: productions for constructing new nodes of a.s.t.

Implementation of array in crème CAraMeL

Semantics – interpreter.ml:

- new value for the environment: Descr_Vector of loc * int * int (VO, LB, UB)
- declaration with initialization to 0 (or 0.)
- evaluation of expression (r-value)
- evaluation of the address (l-value)