Addressing Array Elements

Elements of an array can be accessed quickly if the elements are stored in a block of consecutive locations. If the width of each array element is w, then the ith element of array A begins in location

base + (i – low) x w

where low is the lower bound on the subscript

base is the relative address of the storage allocated for the array.

I.e., base is the relative address of A[low].

The expression can be partially evaluated at compile time if it is rewritten as

i x w + (base – low x w)

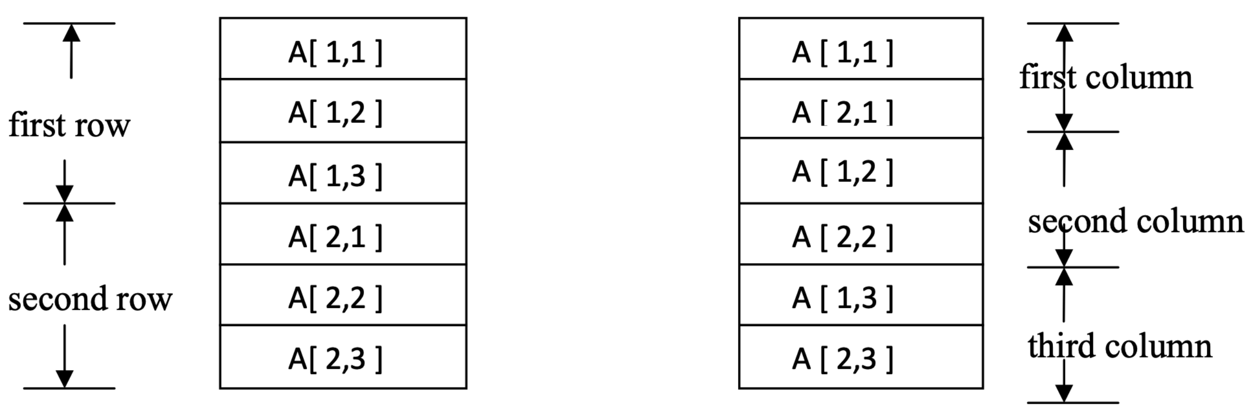
The subexpression ***c = base – low x w*** can be evaluated when the declaration of the array is seen. We assume that c is saved in the symbol table entry for A , so the relative address of A[i] is obtained by simply adding i x w to c.

Address calculation of multi-dimensional arrays:

A two-dimensional array is stored in of the two forms :

1. Row-major (row-by-row)
2. Column-major (column-by-column)

Layouts for a 2 x 3 array



(a) ROW-MAJOR **(b) COLUMN-MAJOR**

In the case of row-major form, the relative address of A[i_1 , i_2] can be calculated by the formula

$$\textbf{\textit{base + ((}}i_1 - low_1) \times n_2 + i_2 - low_2) \times w$$

where, low_1 and low_2 are the lower bounds on the values of i_1 and i_2 and n_2 is the number of values that i_2 can take. That is, if $high_2$ is the upper bound on the value of i_2, then $n_2 = high_2 - low_2 + 1$.

Assuming that i_1 and i_2 are the only values that are known at compile time, we can rewrite the above expression as

$$((i_1 \times n_2) + i_2) \times w + (base - ((low_1 \times n_2) + low_2) \times w)$$

Generalized formula:

The expression generalizes to the following expression for the relative address of $A[i_1,i_2,...,i_k] = ((\ .\ .\ .\ ((i_1n_2 + i_2)\ n_3 + i_3)\ .\ .\ .\ )\ n_k + i_k) \times w + base - ((\ .\ .\ .((low_1n_2 + low_2)n_3 + low_3)\ .\ .\ .)n_k + low_k) \times w$ for all j, $n_j = high_j - low_j + 1$

The Translation Scheme for Addressing Array Elements :

Semantic actions will be added to the grammar :

(1) S → L : =E
(2) E → E +E
(3) E → (E)
(4) E → L
(5) L → Elist]
(6) L → id
(7) Elist → Elist , E
(8) Elist → id [E

We generate a normal assignment if L is a simple name, and an indexed assignment into the location denoted by L otherwise :

(1) S → L : = E

```
{ if L.offset = null then / * L is a simple id */
      gen ( L.place ‘: =’ E.place ) ;
  else
       gen ( L.place ‘ [‘ L.offset ‘ ]’ ‘: =’ E.place)
}
```

(2) $E \rightarrow E_1 + E_2$ { E.place : = newtemp;
gen (E.place ': =' E_1.place ' +' E_2.place) }

(3) $E \rightarrow (E_1)$ { E.place : = E_1.place }

When an array reference L is reduced to E , we want the r-value of L. Therefore we use indexing to obtain the contents of the location L.place [L.offset] :

(4) $E \rightarrow L$

```
{ if L.offset = null then /* L is a simple id* /
        E.place : = L.place
  else begin
        E.place : = newtemp;
        gen ( E.place ': =' L.place ' [' L.offset ']')
end }
```

(5) $L \rightarrow$ Elist]

```
{ L.place : = newtemp;
L.offset : = newtemp;
gen (L.place ': =' c( Elist.array ));
gen (L.offset ': =' Elist.place '*' width (Elist.array)) }
```

(6) $L \rightarrow$ **id**

{ *L.place* := **id**.*place;*
L.offset := **null** }

(7) Elist $\rightarrow$ $Elist_1$, E

```
{ t := newtemp;
m : = Elist1.ndim + 1;
gen ( t ': =' Elist1.place '*' limit (Elist1.array,m));
gen ( t ': =' t '+' E.place);
Elist.array : = Elist1.array;
Elist.place : = t;
Elist.ndim : = m }
```

(8) *Elist* $\rightarrow$ **id** [*E*

{ *Elist.array* : = **id**.*place;*

Elist.place : = *E.place;*
Elist.ndim : = 1 }