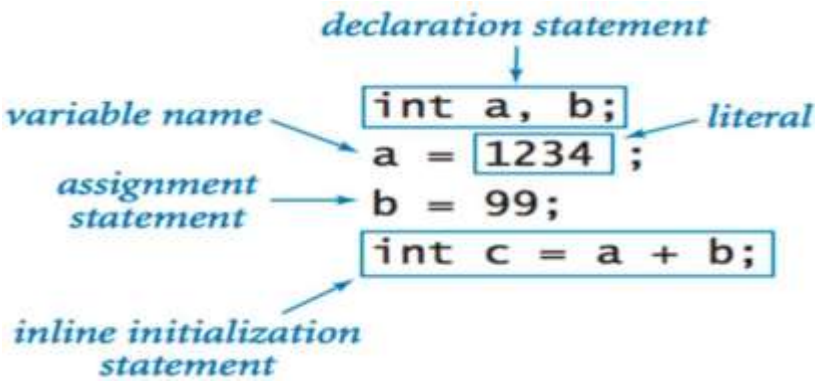
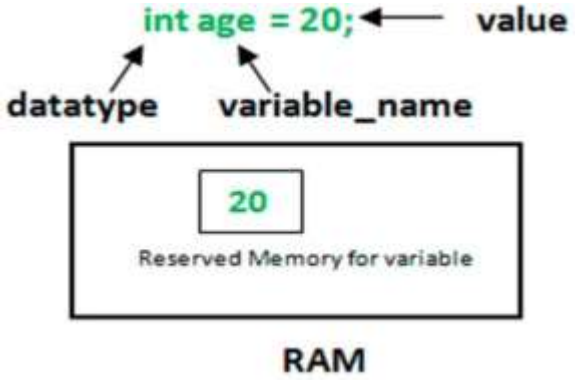
SNS COLLEGE OF TECHNOLOGY

(Autonomous)

COIMBATORE-35

Declarations & Assignments





Declarations

- A symbol table entry is created for every declared name
- Information includes name, type, relative address of storage, etc.
- Relative address consists of an offset
- Types are assigned attributes type and width (size)

Declarations Example

Production	Semantic Rules
P → D	offset := 0
D → D ; D	
D → id : T	enter(id.name, T.type, offset); offset := offset + T.width
T → integer	T.type := integer; T.width := 4
T → real	T.type := real T.width := 8
T → array[num] of T_1	T.type := array(num, T_1.type); T.width := num * T_1.width
T → ↑T	T.type := pointer(T_1.type); T.width := 4

Assignment Statements

Assignment Statements

- Emit (instruction): emit a three address statement to the output
- newtemp: return a new temporary
- lookup(identifier): check if the identifier is in the symbol table.
- Grammar:

  S->id:=E

  E->E1+E2

  E->E1*E2

  E->-E

  E->(E1)

  E->id

Translation scheme to produce three address code for Assignment Statements

Assignment Statements

- *lookup*(*id.name*): returns storage position of *id*.
- *newtemp*(): returns storage position for new temp.

S	$\rightarrow$	$id := E$	$p = lookup(id.name);$
			$if\ p \neq nil\ then\ emit(p\ ':='\ E.place)\ else\ error$
E	$\rightarrow$	$E_1 + E_2$	$E.place = newtemp();$
			$emit(E.place\ ':='\ E_1.place\ '+'\ E_2.place)$
E	$\rightarrow$	$E_1 * E_2$	$E.place = newtemp();$
			$emit(E.place\ ':='\ E_1.place\ '*'\ E_2.place)$
E	$\rightarrow$	$-E_1$	$E.place = newtemp();$
			$emit(E.place\ ':='\ 'uminus'\ E_1.place)$
E	$\rightarrow$	(E_1)	$E.place = E_1.place$
E	$\rightarrow$	id	$p = lookup(id.name);$
			$if\ p \neq nil\ then\ E.place = p\ else\ error$

Reusing Temporary Names

- The code generated by E->E1+E2 has the general form:

  Evaluate E1 into t1

  Evaluate E2 into t2

  t3 := t1 + t2

  - » All temporaries used in E1 are dead after t1 is evaluated
  - » All temporaries used in E2 are dead after t2 is evaluated
  - » T1 and t2 are dead after t3 is assigned
- Temporaries can be managed as a stack:
  - » Keep a counter c, newtemp increases c, when a temporary is used, decreases c.
  - » See the previous example.

Addressing Array Elements

- Arrays are typically stored in block of consecutive locations.
- One-dimensional arrays
  - A: array[low..high] of …
  - the ith elements is at:

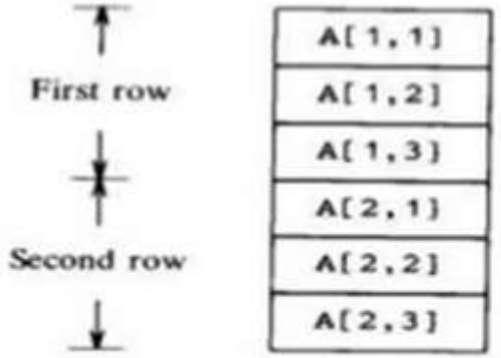
    base + (i-low)*width ➔ i*width + (base – low*width)
- Multi-dimensional arrays:
  - Row major or column major forms
    - Row major: a[1,1], a[1,2], a[1,3], a[2,1], a[2,2], a[2,3]
    - Column major: a[1,1], a[2,1], a[1, 2], a[2, 2],a[1, 3],a[2,3]
  - In raw major form, the address of a[i1, i2] is

    Base+((i1-low1)*(high2-low2+1)+i2-low2)*width

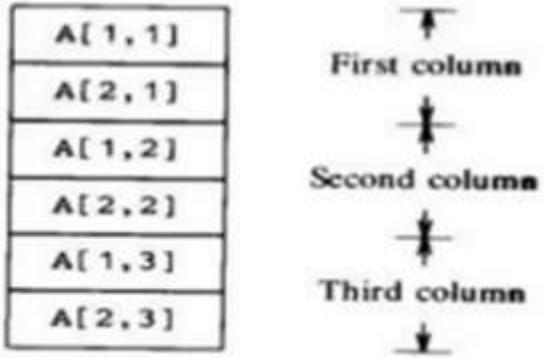
Addressing Array Elements

- In raw major form, the address of a[i1, i2] is

  Base+((i1-low1)*(high2-low2+1)+i2-low2)*width



(a) ROW-MAJOR



(a) COLUMN-MAJOR

Layout for a two dimensional array

Addressing Array Elements

- In general, for any dimensional array, the address of a[i1, i2, i3, …, ik] can be computed as follows:
  - Let highj and lowj be for bounds for ij
  - Let nj = highj – lowj + 1
  - The address is
  - ((…(i1*n2+i2)*n3+…)nk+ik)*width + base – ((…low1*n2+low2)*n3+…)nk+lowk) * width

- When generating code for array references, we need to explicitly compute the address.

Translation Scheme for Array Elements

- Limit(array, j) returns nj=highj-lowj+1
- .place: the temporarys or variables
- .offset: offset from the base, null if not an array reference
- Grammar:

  S->L := E

  E->E+E

  E->(E)

  E->L

  L->Elist]

  L->id

  Elist->Elist, E

  Elist->id[E

Translation Scheme for Array Elements

```
S->L := E {  if L.offset = null then emit(L.place ':=' E.place)
             else emit(L.place'['L.offset ']' ':=' E.place);}
E->E1+E2  {E.place := newtemp;
             emit(E.place ':=' E1.place '+' E2.place);}
E->(E1)  {E.place := E1.place;}
E->L  {if L.offset = null then E.place = L.place
       else {E.place = newtemp;
             emit(E.place ':=' L.place '[' L.offset ']');
            }
       }
L->Elist ] {L.place = newtemp; L.offset = newtemp;
            emit(L.place ':=' c(Elist.array));
            emit(L.offset ':=' Elist.place '*' width(Elist.array);
           }
```

Translation Scheme for Array Elements

```
L->id {L.place = lookup(id.name);
        L.offset = null;
}
Elist->Elist1, E {
   t := newtemp;
   m := Elist1.ndim + 1;
   emit(t ':=' Elist1.place '*'limit(Elist1.array, m));
   emit(t, ':=' t '+' E.place);
   Elist.array = Elist1.array;
   Elist.place := t;
   Elist.ndim := m;
}
Elist->id[E {Elist.array := lookup(id.name);
             Elist.place := E.place
             Elist.ndim := 1;
            }
```

Type Conversions with Assignments

E.place := newtemp;
if E1.type = integer and E2.type = integer then begin
emit(E.place':='E1.place 'int+' E2.place);
E.type: = integer end else if *E1.type = real* and *E2.type = real* then begin
emit (E.place ':='E1.place "real +" *E2.place);*
E.type := real end else if *E1.type = integer* and *E2.type = real* then begin
u := newtemp;
emit(u ':='""inttoreal" *E1.place);*
emit(E.place ':='u "real+" *E2.place);*
E.type:= real end else if *E1.type = real* and *E2.type = integer* then begin
u := newtemp;
emit(u ':='""inttoreal" *E2.place);*
emit(E.place ':='E1.place "real+" u);
E.type: = real end else
E.type:= type error;

Accessing Fields in Records

- The compiler must keep track of both the types and relative addresses of the fields of a record.
- An advantage of keeping this information in symbol-table entries for the field names is that the routine for looking up names in the symbol table can also be used for field names.

T → record L D end { T.type := record(top(tblptr));
T.width := top(offset);
Pop(tblptr); pop(offset) }

L →ε { t:= mktable(nil);
Push(t, tblptr); push (0, offset) }

Accessing Fields in Records

- If *r* is a pointer to the symbol table for a record type, then the type *record(t)* formed by applying the constructor *record* to the pointer was returned as *T.type*

We use the expression ,

p↑.info + 1

- From the operations in this expression it follows that p must be a pointer to a record with a field name info whose type is arithmetic.

Pointer(record(t))

The type of pt is then record(t),from which t can be extracted. The field name info is looked up in the symbol table pointed to by t

Summarization