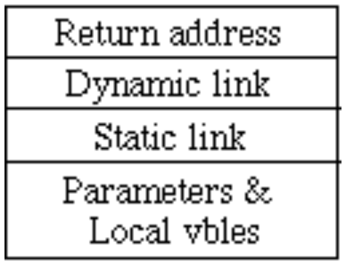
1 Stack-based languages using ALGOL 60/Pascal

Problem during procedure activation

- Static (e.g. scope) vs. dynamic (e.g. return address) environments

Stack reflects dynamic environment. An activation record pushed onto stack each time there is a procedure call, and then popped off after return.



Ex.

```
Program main;
    type array_type = array [1..10] of real;
    var a : integer;
         b : array_type;
    Procedure x (var c : integer; d : array_type);
        var e : array_type;
        procedure y (f : array_type);
            var g : integer;
            begin
                    :
                z(a+c);
                    :
            end; {y}
        begin {x}
             : ..... := b[6].......
            y(e);
             :
        end; {x}
    Procedure z (h : integer);
        var a : array_type;
        begin
                :
            x (h,a);
                :
        end;
    begin {main}
            :
```

```
    x (a,b);
        :
 end. {main}
```

We have the following dynamic calling sequence:

```
Main => x =>  y => z => x => y ....
```

Draw a picture of run-time stack after x calls y 2nd time.

How do we get reference to b in x? to a in y? Where can these variables be held?

Dynamic link (called control link in text) provides pointer to beginning (usually) of caller's activation record.

The static (access) link provides pointer to beginning of activation record of statically containing program unit.

How do find location of variable? Easy in FORTRAN! Not here!

Must keep track of the static nesting level of each variable and procedure.

When access vble or procedure, subtract static nesting level of definition from static nesting level of the use.

Tells how far down static chain to environment of definition.

Example:

```
Name          Level    Name           Level

main            0          y            2
    a           1              f        3
    b           1              g        3
    x           1      z                1
        c       2          h            2
        d       2          a            2
        e       2
```

Notes:

1. Length of static chain from any fixed procedure to main program is always same length (doesn't depend on which activation we are in).
2. Any non-local vble will be found after some fixed number of static links (doesn't depend on which activation we are in).
3. This # of links is a constant determinable at compile time! (Difference between nesting level of call and callee.)

Thus represent identifier references in program as pair:

- <chain position, offset>

Eg: from within y represent d as <1,nx+2> where nx is size of activation record of x before parameters. Similarly a is represented as <2, nmain+1>.