Local Variable Allocation

• Local scalar variables are bound to storage within an activation record instance
• Local variables that are structures are sometimes allocated elsewhere, and only leave their descriptors and a pointer to the storage as part of the activation record

An Example

void sub(float total, int part) {
    int list[5];
    float sum;
    ...
}

<table>
<thead>
<tr>
<th>Local</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>list[4]</td>
</tr>
<tr>
<td></td>
<td>list[3]</td>
</tr>
<tr>
<td></td>
<td>list[2]</td>
</tr>
<tr>
<td></td>
<td>list[1]</td>
</tr>
<tr>
<td>Local</td>
<td>list[0]</td>
</tr>
<tr>
<td>Local</td>
<td>part</td>
</tr>
<tr>
<td>Local</td>
<td>total</td>
</tr>
<tr>
<td>Parameter</td>
<td>Dynamic link</td>
</tr>
<tr>
<td>Dynamic link</td>
<td>Return address</td>
</tr>
</tbody>
</table>
Recursion

- Function recursion means that a function can eventually call itself
- Recursion adds the possibility of multiple simultaneous activations of a subroutine at a given time, with at least one call from outside the subroutine, and one or more recursive calls
- Each activation requires its own activation record instance

An Example

```c
int factorial(int n) {
    if (n <= 1)
        return 1;
    else return (n * factorial(n - 1));
}
void main() {
    int value;
    value = factorial(3);
}
```

How does the stack change?
Implementing nested subroutines

• Some static-scoped languages use stack-dynamic local variables and allow subroutines to be nested
  – FORTRAN 95, Ada, Python, and JavaScript
• Challenge
  – How to access nonlocal variables?

Two-step access process

• Find the activation record instance on the stack where the variable was allocated
  – more challenging and more difficult
• Use the local_offset of the variable to access it
  – local_offset describes the offset from the beginning/bottom of an activation record
Key Observations

- In a given subroutine, only variables that are declared in static ancestor scopes are visible and can be accessed.
- Activation record instances of all static ancestors are always on the stack when variables in them are referenced by a nested subroutine: A subroutine is callable only when all its static ancestors are active.

Finding Activation Record Instance

- Static chaining
  - A new pointer, static link (static scope pointer or access link), is used to point to the bottom of an activation record instance of the static parent.
  - The pointer is used for access to nonlocal variables.
  - Typically, the static link appears below parameters in an activation record.
Finding Activation Record Instance

• A static chain is a chain of static links that connect the activation record instances of all static ancestors for an executing subroutine
• This chain can be used to implement nonlocal variable access

Finding Activation Record Instance

• With static links, finding the correct activation record instance is simple
  – Search the static chain until a static ancestor is found to contain the variable
• However, the implementation can be even simpler
  – Compiler identifies both nonlocal references, and the length of static chain to follow to reach the correct record
Finding Activation Record Instance

- **static_depth** is an integer associated with a static scope that indicates how deeply it is nested in the outermost scope.
- The difference between the static_depth of a nonlocal reference and the static_depth of the variable definition is called **nesting_depth**, or **chain_depth**, of the reference.
- Each reference is represented with an ordered integer pair (chain_offset, local_offset).

An Ada Example [4]

```ada
procedure Main_2 is
  X : Integer;
  procedure Bigsub is
    A, B, C : Integer;
    procedure Sub1 is
      A, D : Integer;
      begin -- of Sub1
        A := B + C; <-----------1
      end; -- of Sub1
    procedure Sub2(X : Integer) is
      B, E : Integer;
      begin -- of Sub3
        Sub1;
        X := B + A; <-------------2
      end; -- of Sub3
    begin -- of Sub2
      Sub3;
      A := D + E; <--------------3
    end; -- of Sub2
    begin -- of Bigsub
      Sub2(?)
    end; -- of Bigsub
    begin
      Bigsub;
    end;
end
```

Main_2 calls Bigsub
Bigsub calls Sub2
Sub2 calls Sub3
Sub3 calls Sub1

What is the static depth for each procedure?
What is the representation of A at points 1, 2, and 3?
procedure Main_2 is
  X : Integer;
  procedure Bigsub is
    A, B, C : Integer;
  begin
    A := B + C;  <-------------1
  end;  -- of Sub1
  procedure Sub3 is
    A, B : Integer;
  begin
    A := B + A;  <---------------2
  end;  -- of Sub2
    A := D + E;  <---------------3
  end;  -- of Bigsub
begin
  Bigsub;
end;  of Main_2 )