Implementing Subprograms with Stack-Dynamic Local Variables

- More complicated because:
  - The compiler must generate code to cause implicit allocation and deallocation of local variables
  - Recursion must be supported (adds the possibility of multiple simultaneous activations of a subprogram)

Typical Activation Record for a Language with Stack-Dynamic Local Variables

<table>
<thead>
<tr>
<th>Local variables</th>
<th>Parameters</th>
<th>Dynamic link</th>
<th>Return address</th>
</tr>
</thead>
</table>

An Example C Function

```c
void sub(float total, int part)
{
    int list[4];
    float sum;
    ...
}
```
An Example C Program Without Recursion

```c
void A(int x) {
    int y;
    C(y);
}
void B(float x) {
    int a, t;
    A(x);
}
void C(int q) {
}
void main() {
    float p;
    B(p);
}
```

Stack Contents For Program

Note that: main calls B
B calls A
A calls C

Implementing Subprograms in ALGOL-like Languages

- The collection of dynamic links in the stack at a given time is called the dynamic chain, or call chain

- Local variables can be accessed by their offset from the beginning of the activation record. This offset is called the local_offset

- The local_offset of a local variable can be determined by the compiler
  - Assuming all stack positions are the same size, the first local variable declared has an offset of three plus the number of parameters, e.g., in main, the local_offset of Y in A is 3

Recursion

- The activation record used in the previous example supports recursion, e.g.

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

<table>
<thead>
<tr>
<th>Functional value</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td></td>
</tr>
<tr>
<td>Dynamic link</td>
<td></td>
</tr>
<tr>
<td>Return address</td>
<td></td>
</tr>
</tbody>
</table>

The Process of Locating a Nonlocal Reference

- Finding the offset is easy
- Finding the correct activation record instance:
  - Static semantic rules guarantee that all nonlocal variables that can be referenced have been allocated in some activation record instance that is on the stack when the reference is made

Nested Subprograms

- Technique 1 - Static Chains
  - A static chain is a chain of static links that connects certain activation record instances
  - The static link in an activation record instance for subprogram A points to one of the activation record instances of A's static parent
  - The static chain from an activation record instance connects it to all of its static ancestors

Static Chains (continued)

- To find the declaration for a reference to a nonlocal variable:
  - You could chase the static chain until the activation record instance (ari) that has the variable is found, searching each ari as it is found, if variable names were stored in the ari
- Def: static_depth is an integer associated with a static scope whose value is the depth of nesting of that scope
**Static Chains (continued)**

- **Definition**: The *chain_offset* or *nesting_depth* of a nonlocal reference is the difference between the static_depth of the reference and that of the scope where it is declared.
- A reference can be represented by the pair:
  - (chain_offset, local_offset)
  - where local_offset is the offset in the activation record of the variable being referenced.

---

**Blocks**

- **Two Methods**:
  1. Treat blocks as parameterless subprograms
     - Use activation records
  2. Allocate locals on top of the ari of the subprogram
     - Must use a different method to access locals
     - A little more work for the compiler writer

---

---

---
Implementing Dynamic Scoping

void C() {
    int x, z;
    x = u + v;
    ...
}
void B() {
    int w, x;
    ...
}
void A() {
    int v, w;
    ...
}
void main() {
    int v, u;
    ...
}

1. Deep Access - nonlocal references are found by searching the activation record instances on the dynamic chain
   - Length of chain cannot be statically determined
   - Every activation record instance must have variable names

2. Shallow Access - put locals in a central place
   - Methods:
     a. One stack for each variable name
     b. Central table with an entry for each variable name
## Using Shallow Access to Implement Dynamic Scoping

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>MAIN_6</td>
<td>MAIN_6</td>
</tr>
<tr>
<td>u</td>
<td>v</td>
</tr>
</tbody>
</table>

(The names in the stack cells indicate the program units of the variable declaration.)