Chapter 9 - Implementing Subprograms

Subprogram linkage: The subprogram call and return operations of a language are together called its subprogram linkage.

Implementing FORTRAN 77 Subprograms:

Call Semantics:

1. Save the execution status of the caller.
2. Carry out the parameter-passing process.
3. Pass the return address.
4. Transfer control to the callee.

Return Semantics:

1. If pass-by-value-result parameters are used, move the current values of those parameters to their corresponding actual parameters.
2. If it is a function, move the functional value to a place the caller can get it.
3. Restore the execution status of the caller.
4. Transfer control back to the caller.

Required Storage:

- Status information of the caller, parameters, return address, and functional value (if it is a function).

Activation Record (AR): The format, or layout, of the noncode part of an executing subprogram.

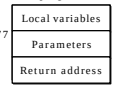
An **activation record instance** is a concrete example of an activation record (the collection of data for a particular subprogram activation)

FORTRAN 77 subprograms can have no more than one activation record instance at any given time since there is no recursion in F77.

The code of all of the program units of a FORTRAN 77 program may reside together in memory, with the data for all units stored together elsewhere.

The alternative is to store all local subprogram data with the subprogram code.

FORTRAN 77 Activation Record:



Linker

The executable program shown in Figure 9.2 (3rd Ed.) is put together not by the compiler but by the **linker** (also called as loader, linker/loader, or link editor), which is part of the operating system. Why not done by the compiler?

1. When the linker is called for the main program, its first task is to find the files that contain the translated subprograms referenced in that program, along with their data areas, and load them into memory.

2. It must determine the size of all COMMON blocks and allocate storage for them.

3. It must set the target addresses of all calls to those subprograms in the main program to the entry addresses of those subprograms.

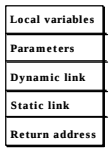
4. The same must be done for all calls to subprograms in the loaded subprograms and all calls to FORTRAN library subprograms.

Implementing Subprograms in ALGOL-like Languages

This is more complicated than implementing FORTRAN 77 subprograms because:

- Parameters are often passed by two methods: by-value and by-reference.
- Local variables are often dynamically allocated.
- Recursion must be supported.
- Static scoping that gives access to nonlocal variables must be supported.

A typical activation record for an ALGOL-like language:



Why local variables are placed in last?

The activation record format is static (known at compile time,) but its size may be dynamic.

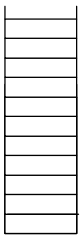
- The **static link** points to the bottom of the activation record instance of an activation of the static parent (used for access to nonlocal vars)
- The **dynamic link** points to the top of an instance of the activation record of the caller. (used for destructing the current AR)

An activation record instance is dynamically created when a subprogram is called.

- 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.

The activation record used in the next example supports recursion.


```
procedure sub(var total:real; part:integer);
var list : array [1..5] of integer;
    sum : real;
begin
  . . .
end;
```



Format of the AR is fixed at compile time, although its size may depend on the call in some languages other than Pascal.

Every procedure activation, creates a new instance of an AR in the stack.

A subprogram is **active** from the time it is called until the time its execution is completed.

```
program MAIN_1;
 var P : real;
 procedure A(X : integer);
  var Y : boolean;
  procedure C(Q : boolean);
   begin { C }
    ... <----------------------3
   end; { C }
  begin { A }
  ... <----------------------2
    C(Y);
  ...
  end; { A }
  procedure B(R : real);
    var S, T : integer;
    begin { B }
    ... <----------------------1
      A(S);
    ...
    end; { B }
begin { MAIN_1 }
  B(P);
end. { MAIN_1 }
```

Note that: MAIN_1 calls B
B calls A and A calls C
In MAIN_1, the local offset of Y in A is 4