Chapter 9 - Implementing Subprograms

Subprogram linkage: The subprogram call and return process in FORTRAN 77 subprograms is part of the subprogram linkage. 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.

Implementing FORTRAN 77 Subprograms: Call Semantics:
1. Save the execution status of the caller.
2. If it is a function, move the functional value (if it is a function).
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.

Static link                stack top
Dynamic link

The executable program shown in Figure 9.2 (HiRes) is put together not by the compiler but by the 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 job is to find the files containing data with the subprogram code. The files are referred to in the 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 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.

Return address

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

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

The activation record used in the next example supports recursion.

The activation record instance of an activation of a subprogram creates a new instance of an activation record. This offset is called the offset from the beginning of the activation record. The collection of all activation records and all calls to a particular subprogram activation is called a call chain.

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.

Implementing Subprograms in ALGOL-like Languages

This is more complicated than implementing FORTRAN 77 subprograms because:
- Parameters are often passed by two methods: value and by-reference.
- Local variables are often dynamically allocated.
- Return value must be supported.
- Static scoping that gives access to nonlocal variables is not supported.

A typical activation record for an ALGOL-like language:

Local variables
Parameters
Dynamic link
Static link
Return address

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 when activation ends).
- 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 local offset gives the number of elements plus the number of parameters.

The activation record used in the next example supports recursion.
program MAIN_1;
var P : real;
procedure A(X : integer);
var Y : boolean;
begin
  C(Y);
end; { C }
begin
  { A }
  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