Compiler design

Programming Languages

Statements

- Simple and compound statements
  - Statements can be *simple* statement
    - Does not contain any embedded statements
    - Read, goto and assignment
  - *compound* statement
    - Contain one or more embeddded statements
    - IF (condition) statement

Type of Statements

1. Computation statements
   - Applying operators on operands to compute a value
   - E.g., assignment statement
2. Sequence-control statements
   - Control flows automatically from one statement to the next
   - *goto, break, call, return deliberately alters the flows of control*
3. Structural statements
   - END serve to group simple statements into structures
4. Declaration statements
   - Does not produce executable code
   - Their semantic effect is to inform the compiler about the attributes of names encountered in a program
5. Input and output statements
   - Mostly implemented by library subroutines

Program units

- FORTRAN
  - Consists of one main program and zero or more subprograms
- ALGOL
  - Block structured language- allows nesting of blocks
  - Uses keyword **begin** and **end**
- PL/I
  - Uses characteristics of ALGOL and FORTRAN
  - Consists of a set of external procedures which are can be compiled separately
  - Has blocks and internal procedures

Data environments

- Consider A:=B+1 in a program
- How is the value of B determined?
- This identifier may appear anywhere in a program
  - These locations may be local or global
- In each case different rules apply to determined the value of B
- The association of identifiers with names they currently denote is called as the Environment of a statement, block, or subprogram

Binding identifiers to Names

- In order to be able to translate a program to machine code we must associate with each mention of an identifier in the program a *location* which this identifier represents
- The association is a Two-Stage process
  1. Determine the name represented by the identifier- (*binding the identifier to a name*)
  2. Bind the name to a location
- Binding of names may be *static* or *dynamic*

Binding identifiers to Names

- Binding of names may be *static* or *dynamic*
- *Static*
  - Binding of an identifier depends only on its position in the program
  - Easier for the compiler to determine the binding
  - *FORTRAN, ALGOL, PL/I*
- *Dynamic*
  - The name denoted by an identifier may not be known until run time
  - SNOBOL, LISP

Scope Rules

- The scope of a name is its portion of the program over which it may be used or may be accessible
- Scope Rules can be *static* or *dynamic*
  - *Static scope rules define the scope of a name in terms of the syntactic structure of the program*
    - *ALGOL, PL/I*
  - *Dynamic scope rules*
    - The region of a program over which a name is valid can vary while the program is running
    - SNOBOL, LISP
      - It is required that a name A refer to the declaration of A in the most recently initiated, but not yet terminated block or procedure having a declaration of A

Parameter Transmission

- The ability to define and call procedures is a great asset in a programming language.
- Procedures
  - Permit modular design of programs, by allowing large tasks to be broken into smaller units
  - Permit economy in the size of programs and in the total programming effort
  - Add extensibility to a language, since operators can be defined in terms of procedures, which can then be used as functions within expressions
- Method of transmission of information to and from procedures must be defined and established

Parameters

- Transmission can be implemented using global variables
- Transmission can be implemented using parameters
- A distinction need to be made between the definition and its use
- Actual parameters and Formal parameters

- Three common methods of passing parameters
  - Call-by-reference
  - Call-by-value
  - Call-by-name

Call-by-reference

- The calling program passes to the subroutine a pointer to the r-value of each actual parameter
- The value of the pointer is put in a known place determined by the language implementation
- If actual parameter is a simple variable having an l-value then that l-value is passed
- If actual parameter is a constant 2 or an expression A+B, then it has no l-value
  - The expression is evaluated in a new location and the l-value of that location is passed

Call-by-value and call-by-name

- The actual parameters are evaluated and their r-values are passed to the subroutine
- Call-by-name
  - A third mechanism was used in early programming languages like ALGOL 60
  - The basic idea is to leave the actual parameters unevaluated until they are needed and
  - To evaluate them anew each time they are needed
  - Implementation is to pass to the called procedure parameterless subroutine called **thunks**
    - These can evaluate the l-value and r-value of actual parameters