Compiler Design

Introduction III

Data elements

- One of the basic building block of programming languages is the set of primitive data elements
  - to which operators can be applied
  - And out of which complex data structures can be built.

Elementary data types

1. Numerical data
2. Logical data (boolean type)
3. Character data
4. Pointers
5. Labels

Identifiers and Names

- Each programming language manipulates and uses objects, which are data elements(int or reals)
- The computer is viewed as consisting of an abstract store, consisting of cells in which values can be kept.
- Each cells has a name- which is a variable
- Each name is denoted by an identifier-which is a string of characters
- The set of legal identifiers varies from language to language
- Each name possesses a value and attributes
- The attribute of a name determine
  - the possible values that the name may have,
  - the operations that may be applied,
  - and the effect of these operations

Attributes

- The attribute of a name determine its properties
- Its type
- Its scope

Declaration

- The attribute of a name are determined implicitly, explicitly or by default
- In FORTRAN any identifier beginning with I,J,…,N is implicitly an integer
- An attribute of a name cab be set by means of an explicit declaration
  - PL/I can have combinations of four kinds of attributes
    - Mode
    - Scale
    - Base
    - Precision
      - DECLARE A DECIMAL FLOAT(10)
      - declares **A** to be of scale floating point, in the decimal base and precision 10 decimal digits

Binding Attributes to Names

- The act of associating attributes to a name is referred as binding the attributes to the name
- Most binding of attributes is done during compilation- ***static binding***
- Binding attributes at run time is called ***dynamic binding*** (SNOBOL)
- If attributes are bound to a name by a declaration, then the attributes are fixed and cannot change at run time
  - PASCAL- hence extensive type-checking is done at compile time
  - APL- dynamic binding of attributes to names is permitted.
    - It is possible to use a name as an integer in one statement and in the next as an array

Binding Attributes to Names

- Hence in dynamic binding, operations on data may involve a check to determine what is the current type of the name.
- In this case all names must have a descriptors as part of their values
- The implementation of operations often involves a subroutine call
  - first action is to examine the descriptors and determine how operators are applied

Data structures

- On the logical level, data structures are a
  - A set of primitives data elements
  - other data structures
  - Set of structrural relations among its components
- Lists
  - Operations are insertion, deletion or substitution
- Trees
  - One element r is called the root of T
  - The remaining nodes can be partitioned into k>=0 sub tree $T_1, T_2, .. T_k$
    - The root $\mathbf{T_i}$ is a child of **r**

Arrays

- Fixed-size array one-dimensional arrays
- Fixed-size multi-dimensional arrays
- Adjustable arrays

Record structures

- Logically a record structure is a tree, with the fields of the record being the children of the root, the subfields being the children of these, and so on

Character Strings

One dimensional arrays whose elements are characters

List structures

Formed from elements which are records with two fields, called Car and CDR

List structures

- Formed from elements which are records with two fields, called Car and CDR
- Example, the linear list A, B, C can be created by using three records.
- CAR of each contains A,B,C
- CDR of the first points to the second and CDR of the second points to the third, CDR of the third points to null
- Use in LISP

Stacks

- A linear list where operation is done only at the beginning or top end.

Operators

- Arithmetic operators
  - Unary and binary operators
- Arithmetic Expressions
  - A single data reference is an expression
  - Data reference
    1. Identifier, constant, array element
    2. If θ is a binary infix operator and X and Y are expressions, then XθY is an expression
    3. If θ is a unary prefix operator and X is an expressions, then θ X is an expression
    4. If E is an expression, the (E) is an expression

Operators

- Relational operators
  - Six common relational operators
- Logical operators
- String Operators

Associativity and precedence

- The order in which operators are applied in an expression depends on the associativity and precedence of a programming language
- a + b * c
- It can be group in two ways and depends on the precedence level to indicate which operands are allowed to group their operands first
- Normally * has higher precedence than +
- Associativity is not the same for all languages
- ALGOL evaluates all binary operators left-associativity
- FORTRAN allows the compiler designer to choose
- Different operators on the same precedence level are treated equally, and associated left to right

Algebraic properties of Operators

- Many operators obey certain algebraic laws which makes some simplification of expression by the compiler possible
  - Enable to produce efficient object code
- Addition and multiplication are
  - commutative
    - A + B=B + A for any expression of A and B
  - Associative
    - (A + B) + C=A + (B + C)
- Multiplication distributes over addition
  - A * (B + C)= A * B + A * C
- However on a real computer only commutative laws holds
  - Others might cause overflows and loss of significant digits

Other Operators

- Conditionals, A=if B then C else D in ALGOL

  A=(B>0)?C:D
- Selectors, e.g., subscripting operator (use in arrays)
- Operators like Subscripting and conditionals have a different syntax
- Conditionals often treated as ternary operators
- Subscripting are *Variadic* or *Polyadic*- it may take any number of operands

Coercion of types

- If a programming language permits operators to have operands with different types, the langauge must provide rules to specify what the type of the result is
- The translation of the operator includes any necessary conversions from one type to another
  - this change is called a **coercion**

Implementation of operators

- Since many of the common operators on primitive data types are implemented by corresponding machine operations
  - Uses arithmetic operators on integers and reals, and logical on boolean values
- Relational operators are implemented by comparison instructions and jumps
- Others may needs calls to subroutines

Assignment

- Various forms of assignment operator

ALGOL	A:=B
APL	A←B
BASIC	LET A=B
COBOL	MOVE B TO A
FORTRAN	A=B
C	A=B

l- and r- values

- The location and values represented by a name are two distinct concepts
- A=B
  - Means “put the value of b in the location denoted by A”
- We understand by the position of A and B in the statement
- The position of A on the left means its location not value
- The position of B on the right means its value not position
- Hence we refer to the values associated with a name as its **r-values** (r stands for right)
- The location denoted by a name as its **l-value**(l stands for left)
- An expression consisting of a single name has the same **l-value** and **r-value** as that name

l- and r- values

1. Every name has an l-value, the location reserved for its value
2. A is an array name, the l-value of A[i] is the location or locations reserved for the ith element of an array.
   - The r-value of A[i] is the value stored there
3. The constant 2 has an r-value but no l-value
4. If Ptr is a pointer, its r-value is the location to which Ptr points and its l-value is the location in which the value of Ptr itself is stored

- Interesting exceptions to these generalities are BLISS and ALGOL 68
- BLISS: A name always denote l-value, to produce the r-value the . Operator is used
  - A←.B + .C
- ALGOL 68: makes a distinction by using a mode which begin with the word **ref**
  - an integer variable has a mode **ref int**–refer to an object of mode **int**
  - In an integer assignment, the left side must be an object of mode **ref int,** the right side must be an object of mode **int** or converted to mode **int (**coersion takes place called as **dereferencing)**

Implementation of Assignment

- One of the many ways in which the compiler can implement the assignment A:=B is the following:

1. The compiler generates code to compute the r-value of B into some register r
2. If the data types of A and B are incompatible, the compiler generates code to convert the r-value of B to a type appropriate for A.
3. If necessary the compiler generates code to detrmine the l-value of A.
4. Finally the computer generates code to store the contents of registers r into the l-value of A.

Implementation of Assignment

- Consider a computer whose memory units is four bytes per word, X andY are integer array in fixed location, indices start at zero, each element require one word.
- **X[I]:=Y[J]**
- The r-value ofY[J] can be computed by a sequence
  - LOAD J,r_2
  - MULT #4,r_2
  - LOAD Y(r2),r_1
  - This brings the r-value ofY[J]TO REGISTER r_1
- $_{\text{To}}$ compute the l-value of X[I], we compute the base of an array X by a sequence
  - LOAD I, r_2
  - MULT #4,r_2

Implementation of Assignment

- To store X[I] execute a STORE instruction with address equal to the base of array X, indexed by what is computed in register 2.
- the entire sequence for **X[I]:=Y[J]** is
  - LOAD J,r_2
  - MULT #4,r_2
  - LOAD Y(r2),r_1
  - LOAD I, r_2
  - MULT #4,r_2
  - STORE r_1, X(r_2)

Assignment as an operator

- Most languages as C, ALGOL 68, BLISS treat the assignment operator as any other binary infix operator
  - Giving the lowest precedence
  - In C
  - A=(B=C+D)+(E=F+G)
  - Interpreted as
  - B=C+D
  - E=F+G
  - A=B+E