Compiler design

Introduction II

Compiler writing tools

- A number of tools has been developed specifically to help construct compilers.
- These tools range from scanner to parser generators to complex systems
  - Compiler-compilers, Compiler-generators, Translator-writing systems
  - The input specification for these systems are
    1. The description of the lexical and syntactic structure of the source language
    2. The description of what output is to be generated for each source language construct
    3. A description of the target machine

Compiler-Construction tools

- Some commonly used tools
- **Scanner generator-** produces lexical analyzer from a regular expression description of tokens
- **Parser generator-** produce syntax analyzers from a grammatical description of a programming language
- **Syntax-directed translation engines-** produce collections of routines for walking the parse tree and generating the intermediate code
- **Code-generator generators**-produce a code generator from a collection of rules for translating each operation of the intermediate language into machine language for a target machine
- **Data-flow analysis engines**-facilitate the gathering of information about how values are transmitted from one part of the program to another.(a key part in code optimization)
- **Compiler-construction toolkits**-provide an integrated set of routines for constructing various phases of a compiler

High level programming languages

- A programming language is a notation in which people can communicate algorithms to computers and to one another.
- Some of the features which makes HLL more preferable to Assembly language or machine language
  - Ease of understanding
  - Naturalness
  - Portability
  - Efficiency of use

Definition of programming languages

- notation with the following features used to define programming languages

1. A language designer could simply and clearly express what programs are valid and what these programs means
2. A programmer could determine what programs he could write and what they do
3. A compiler designer could determine what source programs a complier should accept and what object code the compiler should produce for them.

syntax

- A program in any language can be viewed as a string of characters chosen from some set, or alphabet, of characters
- The rules that tells us whether a string is valid or not are called syntax of the language
- Regular expressions and context free grammars are useful for specifying the syntax of programming languages
  - Also aids in construction of compilers

semantics

- The rules that gives meaning to a program are called the **semantics** of the programming language
- The semantics of a programming language is much harder to specify than the syntax
- No completely satisfactory means for specifying semantics in a way that helps construct a correct compiler for the language has been found.

Approaches to the specification of semantics of languages

1. Interpretive (or operational) semantics
   - A machine language has its semantics defined by the computer itself.
     - -it means exactly what the computer does
   - The interpretive approach is to postulate an abstract machine and provide rules for executing programs on this abstract machine
   - These rules define the meaning of programs

Approaches to the specification of semantics of languages

2. Translation
   - The translation of assembly language into machine language is direct and comprehensible-forms a useful semantics specification for an assembly language
   - Such an approach can be applied to other higher level languages
   - Rules can be associated with each program a sentence in a language whose semantics we already understand
     - Using mathematical lambda calculus or a specific machine language

Approaches to the specification of semantics of languages

3. Axiomatic definition
   - Rules can be defined that relate the data before and after execution of each program construct.
     - Useful to define semantics for a part rather than for all of a language.
4. Extensible definition
   - In this approach we define certain primitive operations and define the meaning of the language in terms of these primitives
5. Mathematical (or denotational) semantics
   - Mathematical objects corresponding to programs are defined, and rules are given translating programs to these abstract objects.

The Hierarchical structure of programming languages

- Since none of these methods has gained universal acceptance, terminology of common choices for the meaning of common programming constructs are introduced.
- Hence the definition of programming languages will be a series of choices made by the language designer
- A programming language is a notation for specifying a sequence of operations to be carried out on data objects
- Both data and objects are grouped together into a hierarchy of program elements. (diagram)
- Each constructs in a programming language has both a logical meaning and an implementation
- E.g., names denoting real-logical meaning denote placeholders for real numbers, implementation is its representation inside the computer(denote a word of memory were sequence of its is kept.

The lexical and syntactic structure of a language

- Consider a string A * B + C
  - Five symbols that can be group together in various ways
  1. Single identifier
  2. Three identifier separated by operators and interpreted as (A * B)+C or as A*(B+C)
     - Which of these two is correct depends on the language definition
- The lexical structure of a language determines what groups of symbols are to be treated as identifiers and operators.
- The syntactic structure determines how these lexical constructs are to be grouped together onto a larger structures called syntactic categories.

Alphabets

- The set of symbols used in a programming language is called its categories set or alphabets
- Machine language has an alphabet only of 1 and 0
- Alphabet sizes of some common languages

Language	Alphabet size	Alphabetic characters	Numeric characters	others
ALGOL 60	114	52	10	52
COBOL	51	26	10	15
FORTRAN	47	26	10	11
PL/I (60 CHARACTERS)	60	29	10	21

Tokens

- The string representing a program can be partitioned at the lowest level into a sequence of substrings called *tokens*
- Each tokens is a sequence of characters whose significance is posseed collectively rather than individually.
- Most languages treat the following as tokens
  1. Constants, e.g., 1,2,3,4.5E6
  2. Identifiers, e.g., A, NUM1, X1,X2
  3. Operators symbols, e.g., +,-,*,:=,.EQ.
  4. Keywords, e.g., IF, GOTO, SUBROUTINE
  5. Punctuation symbols, e.g., parentheses (), brackets {}, comma , and semicolon ;

Higher Level Constructs

- Syntactic structures are grouping of tokens
- Expressions, which are sequences of operators and operands tokens, e.g., A+2.0*B
- Expression together with assignment operator, punctuation symbols, and keywords, form the building blocks for the higher-level constructs such as statements, blocks, and programs.

Lexical convention

- The physical presentation of programs can affect the difficulty of lexical analysis
- FORTRAN require certain statement elements to appear in **restricted positions** on the input line.(fixed position)
- The trend in HLL design is towards free-format input, allowing statements to appear anywhere in the in the input line
  - Help to avoid syntax errors due to improper positioning of statements
- Next is blanks
  - They are not significance in FORTRAN and AKGOL 68
    - -this complicates the task of identifying tokens
  - In other languages blanks are use as tokens separators
    - Their presence and absence affect the syntactic structure placed on a program