Fundamental Concepts of Programming Languages

Implementation of PLs
Lecture 04

conf. dr. ing. Ciprian-Bogdan Chirila

University Politehnica Timisoara
Department of Computing and Information Technology

1. The implementation of PLs
2. Interpretation
3. Translation
4. Comparisons
5. The compiling process
6. Compiler structure
7. Analysis and synthesis
8. Compiling the assignment instruction

Implementation of PLs

- All computers execute low level programs written in machine language
- In order to execute high level languages two methods are used:
  - interpretation
  - translation

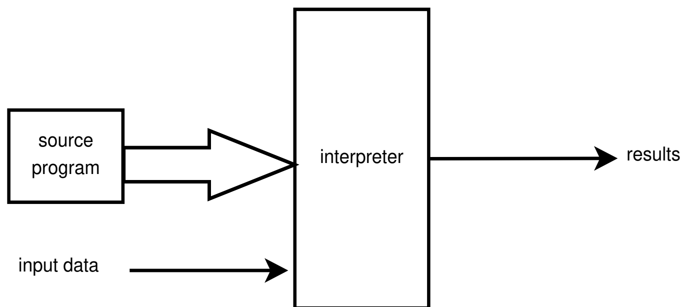
1 The implementation of PLs
2 Interpretation
3 Translation
4 Comparisons
5 The compiling process
6 Compiler structure
7 Analysis and synthesis
8 Compiling the assignment instruction

Interpretation

- Means executing directly the high level instructions
- Each high level instruction consists in a sequence of machine instructions
- Program execution is done by an interpreter
  - Reads the high level instructions
  - Decodes them
  - Executes the machine instructions sequence

Interpreter working cycle

- Read next instruction
- Decode instruction
- Execute corresponding machine instructions



Interpretation

- follows the von Neumann normal cycle
- is a simulation on a regular computer of a high level language computer

1 The implementation of PLs

2 Interpretation

3 Translation

4 Comparisons

5 The compiling process

6 Compiler structure

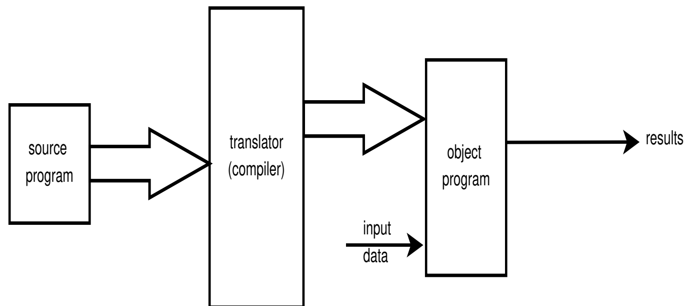
7 Analysis and synthesis

8 Compiling the assignment instruction

Translation

- Before execution the program is translated from high level language in machine code
- The process is highly complex
- Done in several steps
- Done by several specialized software
  - Preprocessors
  - Compilers
  - Assemblers
  - Link editors
- The result is the object program, machine code to be executed

Implementation by translation



- Compiler - compilation

Translation with interpretation

- Translation into object program
  - not machine code
  - but intermediary code (abstract machine)
- Interpretation of the intermediary code
- present in Java and C#

The Java approach

- Source program compiles into bytecode
- Bytecode instruction sequence for the Java virtual machine (JVM)
- Bytecode can be transferred to any machine having an interpreter
- Advantages
  - High Portability
  - Platform independence
- Drawback
  - Increased interpretation time
- Compromise solution
  - JIT compiler (just in time) - bytecode to machine code

The C# approach

- Compilation result is a pseudocode called MicroSoft intermediate language (MSIL)
- MSIL
  - is a portable assembly language
  - Needs the Common Language Runtime (CLR) to be converted in machine code
  - CLR activates a JIT compiler to convert MSIL code into machine code as needed
- fast
- portable

1. The implementation of PLs
2. Interpretation
3. Translation
4. Comparisons
5. The compiling process
6. Compiler structure
7. Analysis and synthesis
8. Compiling the assignment instruction

Time comparison

- Compiled programs are faster then interpreted programs
- Instruction interpreting implies instruction translation
- Object program
    - Compiled once
    - Run several times without translation
- Interpretation time becomes critical when the application is run several times

Space comparison

- Interpreting takes less memory space
- Compilation involves replacing each high level instruction with the sequence of machine instructions

1 The implementation of PLs
2 Interpretation
3 Translation
4 Comparisons
5 The compiling process
6 Compiler structure
7 Analysis and synthesis
8 Compiling the assignment instruction

The compiling process

- Basic compiler functionalities
  - Source program analysis
  - Destination program synthesis
- The compiler parts
  - Analysis part
    - breaks the program into basic components
    - Creates an intermediate representation
  - Synthesis part
    - Builds the destination program from the intermediate representation

1 The implementation of PLs
2 Interpretation
3 Translation
4 Comparisons
5 The compiling process
6 Compiler structure
7 Analysis and synthesis
8 Compiling the assignment instruction

Compiler structure

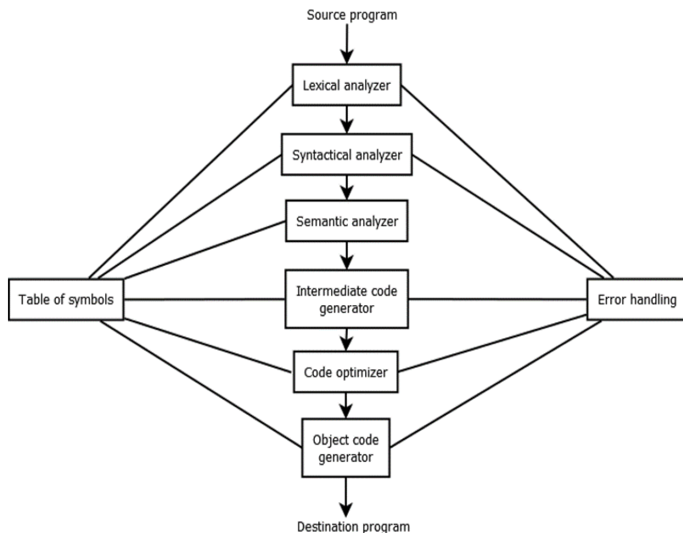


Table of symbols

- Basic compiler tasks
  - to manage identifiers
  - to gather information about their attributes
    - Constants
    - Variables (domain, type)
    - Functions (number, type, order, transmission type for parameters)
- Is a data structure with a record for each identifier
- Must allow fast search of identifiers
- Adding a record in lexical or syntactical analysis
- Auxiliary information are added during the analysis process
- Information is used
  - To check semantic actions
  - To generate the correct object code

Error handling

- Errors can be discovered in the first phase of the compilation
- Different reactions
  - To stop at first error, to correct it and to recompile from the start
  - To handle the errors such as to continue compilation, to detect also other errors and to correct them globally
    - Returning from error
- Most errors are detected in syntactical and semantic analysis phases

1 The implementation of PLs
2 Interpretation
3 Translation
4 Comparisons
5 The compiling process
6 Compiler structure
7 Analysis and synthesis
8 Compiling the assignment instruction

Analysis phases

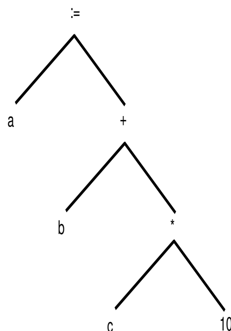
- Lexical or linear analysis
- Syntactical or hierarchy analysis
- Semantic or contextual analysis

Lexical or linear analysis

- The program source
  - is a string of characters
  - is read from left to right
  - is grouped in lexical symbols sequences of characters with specific semantic
- Example a:=b+c*10
  - Identifiers: a, b, c
  - Operators: :=, +, *
  - Integers: 10
- White spaces are ignored

Syntactical or hierarchy analysis

- Symbols are grouped in greater collections
  - Expressions, declarations, instructions
- Syntactic tree
  - Each node represents an operation
  - The sons represent the arguments



Syntactical or hierarchy analysis

- The hierarchical structure is expressed by recursive rules
  - Recursive rules for defining
    - expressions
    - instructions
- Splitting lexical and syntactical analysis is arbitrary
  - To simplify the overall task
  - Numbers, strings, identifiers, punctuation are lexical symbols
  - Expressions, instructions, declarations are syntactical constructions

Semantic or contextual analysis

- Verifications that relate to the meaning of the program
- Source program contextual restrictions
- Gathers type information for the code generation
- Identifies operators, operand and instructions using the hierarchical structure
- Type checking verifies whether each operator has the right operands
  - E.g. a real number can not be used to index a table
- Domain analysis verifies that each identifier is used in its own visibility domain

Synthesis phases

- Intermediate code generation
- Code optimization
- Object code generation

Intermediate code generation

- Intermediate representation of the source text is done after lexical and syntactic analysis
- Can be seen as a program for an abstract computer
- There are several forms of intermediate representations
- 3 address code
  - looks like an assembly language for a computer
  - Each memory location plays the role of a register

The 3 address code

- Sequence of instructions
- At most 3 operands
- Each instruction has at most one operator together with the assignment
- The compiler needs to decide on the order of the operators priority)
- In order to keep computed values in each instruction the compiler must generate temporary variables
  - With no relation with the source text
- There can be instructions with less than 3 operands

Code optimization

- The purpose is to optimize the intermediate code to make fast machine code
- To eliminate
  - Redundancies
  - Useless calculation, variables
- Compilers with optimizations
  - The amount of time consumed for optimization is large
- Simple optimizations
  - Good code efficiency
  - Do not slow down too much compilation

Object code generation

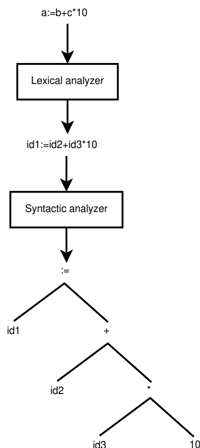
- Is the final phase of the compiler
- Generated object code can be
  - Locatable machine code
  - Virtual code
- To translate intermediate code into machine code
- To select and allocate memory cells for the program variables
- To choose and implement the best variable access techniques using the hardware addressing facilities: indexing, indirection etc
- To allocate registers for computation and for temporary storage of the intermediate results

1 The implementation of PLs
2 Interpretation
3 Translation
4 Comparisons
5 The compiling process
6 Compiler structure
7 Analysis and synthesis
8 Compiling the assignment instruction

Compiling an assignment instruction

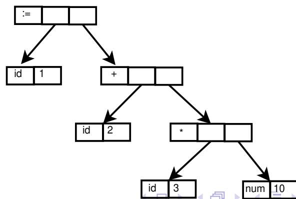
a:=b+c*10

a, b, c - real type variables

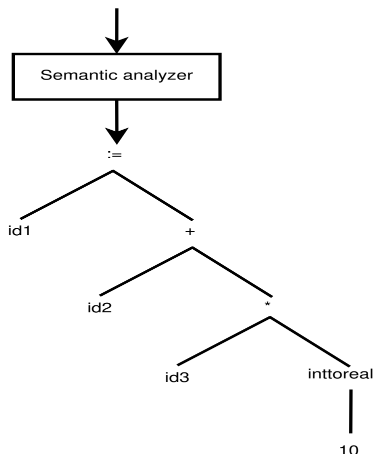


TS

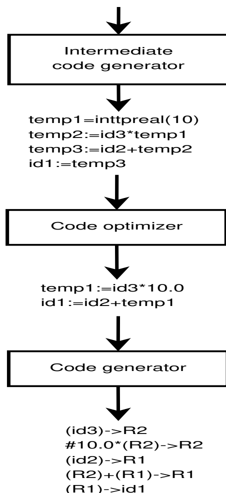
1	a	....
2	b	....
3	c	....
4		....



Compiling an assignment instruction



Compiling an assignment instruction



Bibliography

1. Brian Kernighan, Dennis Ritchie, C Programming Language, second edition, Prentice Hall, 1978.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Horia Ciocarlie – Universul limbajelor de programare, editia 2-a, editura Orizonturi Universitare, Timisoara, 2013.