Compilers

Introduction

محاضرة مادة المترجمات

م . شيماء طه احمد

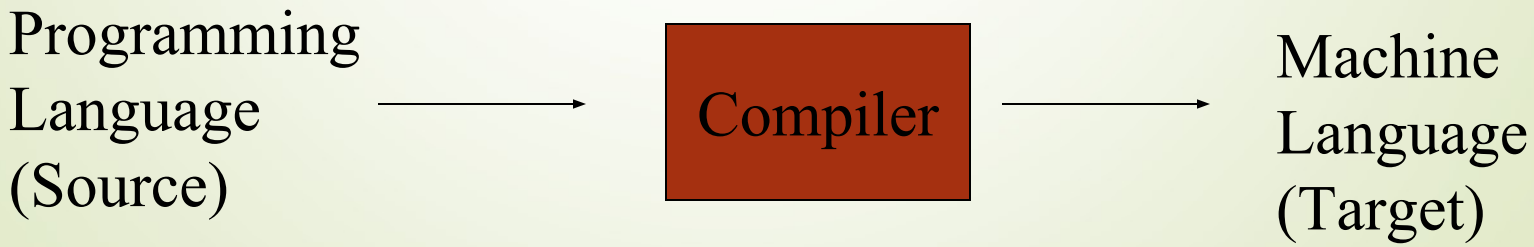
للمرحلة الثالثة حاسبات /كلية التربية الاساسية

Outlines

- Overview and History 1.1
- ?What Do Compilers Do 1.2
- The Structure of a Compiler 1.3
- The Syntax and Semantics of 1.4 Programming Languages
- Compiler Design and Programming 1.5 Language Design
- Compiler Classifications 1.6
- Influences on Computer Design 1.7

Overview and History

- .Compilers are fundamental to modern computing
- They act as translators, transforming human-oriented programming languages .into computer-oriented machine languages



Overview and History (Cont'd)

- The first real compiler
  - FORTRAN compilers of the late 1950s
  - person-years to build 18
- Compiler technology is more broadly applicable and has been employed in rather unexpected areas.
  - Text-formatting languages, like nroff and troff; preprocessor packages like eqn, tbl, pic
  - Silicon compiler for the creation of VLSI circuits
  - Command languages of OS
  - Query languages of Database systems

?What Do Compilers Do

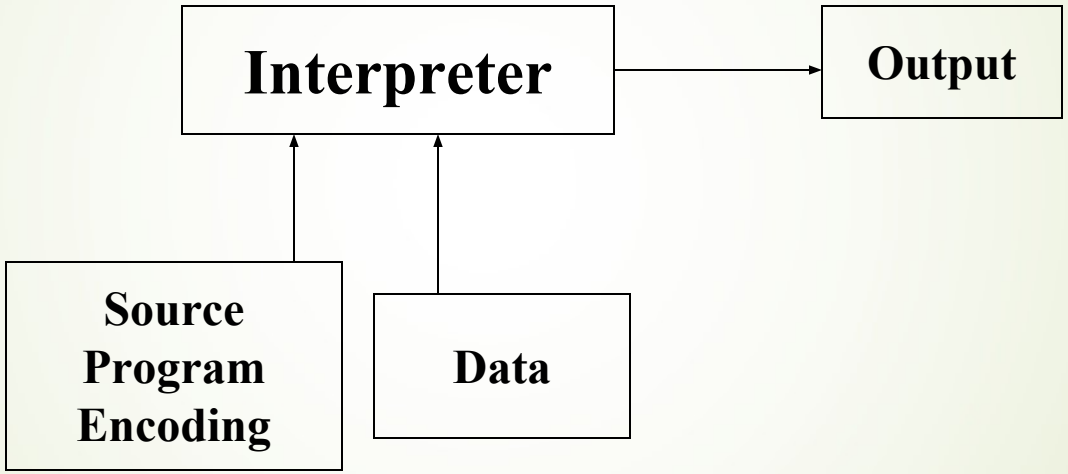
- Compilers may be distinguished according to the kind of target code they generate:
  - Pure Machine Code
  - Augmented Machine Code
  - Virtual Machine Code
    - JVM, P-code

What Do Compilers Do? (Cont'd)

- Another way that compilers differ from one another is in the format of the target machine code they generate
  - Assembly Language Format
  - Relocatable Binary Format
    - A linkage step is required
  - Memory-Image (Load-and-Go) Format

What Do Compilers Do? (Cont'd)

- Another kind of language processor, called an ***interpreter***, differs from a compiler in that it executes programs without explicitly performing a translation



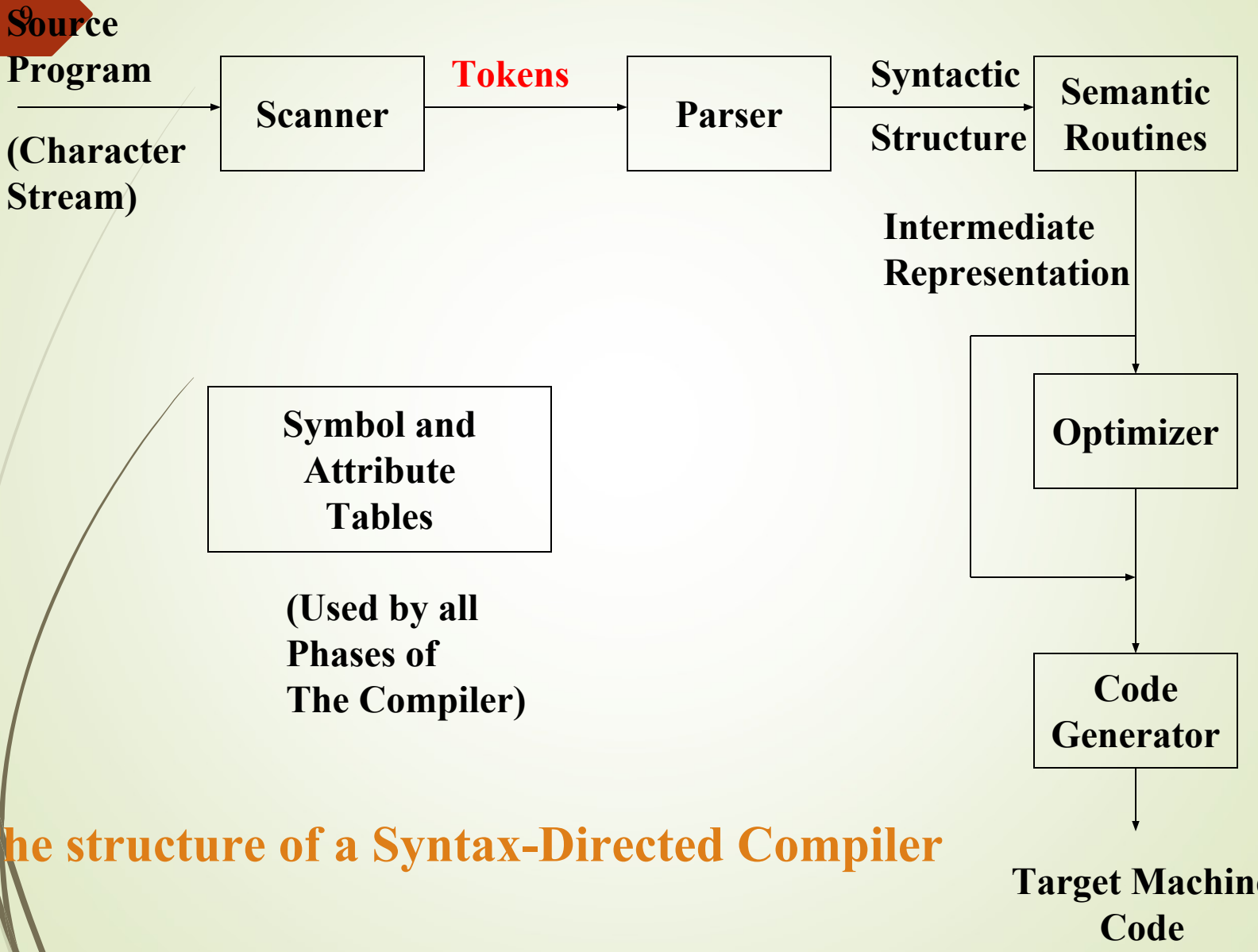
- Advantages and Disadvantages of an interpreter

See page 6 & 7

The Structure of a Compiler

- Any compiler must perform two major tasks
  - ***Analysis*** of the source program
  - ***Synthesis*** of a machine-language program

The Structure of a Compiler (Cont'd)



The structure of a Syntax-Directed Compiler

The Structure of a Compiler (Cont'd)

- Scanner
  - The scanner begins the analysis of the source program by reading the input, character by character, and grouping characters into individual words and symbols (**tokens**)
  - The **tokens** are encoded and then are fed to the parser for syntactic analysis
  - .For details, see the bottom of page 8 and page 9
- Scanner generators

The Structure of a Compiler (Cont'd)

- Parser
  - Given a formal syntax specification (typically as a context-free [CFG] grammar), the parse reads tokens and groups them into units as specified by the productions of the CFG being used.
  - While parsing, the parser verifies correct syntax, and if a syntax error is found, it issues a suitable diagnostic.
  - As syntactic structure is recognized, the parser either calls corresponding semantic routines directly or builds a syntax tree.

The Structure of a Compiler (Cont'd)

- Semantic Routines
  - Perform two functions
    - Check the static semantics of each construct
    - Do the actual translation
  - The heart of a compiler
- Optimizer
  - The IR code generated by the semantic routines is analyzed and transformed into functionally equivalent but improved IR code.
  - This phase can be very complex and slow
  - Peephole optimization

The Structure of a Compiler (Cont'd)

- One-pass compiler
  - No optimization is required
  - To merge code generation with semantic routines and eliminate the use of an IR
- Compiler writing tools
  - Compiler generators or compiler-compilers
    - E.g. scanner and parser generators

Compiler Design and Programming Language Design

- An interesting aspect is how programming language design and compiler design influence one another.
- Programming languages that are easy to compile have many advantages
  - See the 2nd paragraph of page 16.

Compiler Design and Programming Language Design (Cont'd)

- Languages such as Snobol and APL are usually considered noncompilable
- What attributes must be found in a programming language to allow compilation?
  - Can the scope and binding of each identifier reference be determined before execution begins
  - Can the type of object be determined before execution begins?
  - Can existing program text be changed or added to during execution?

Compiler Classifications

- Diagnostic compilers
- Optimizing compilers
- Retargetable compiler

Compilers

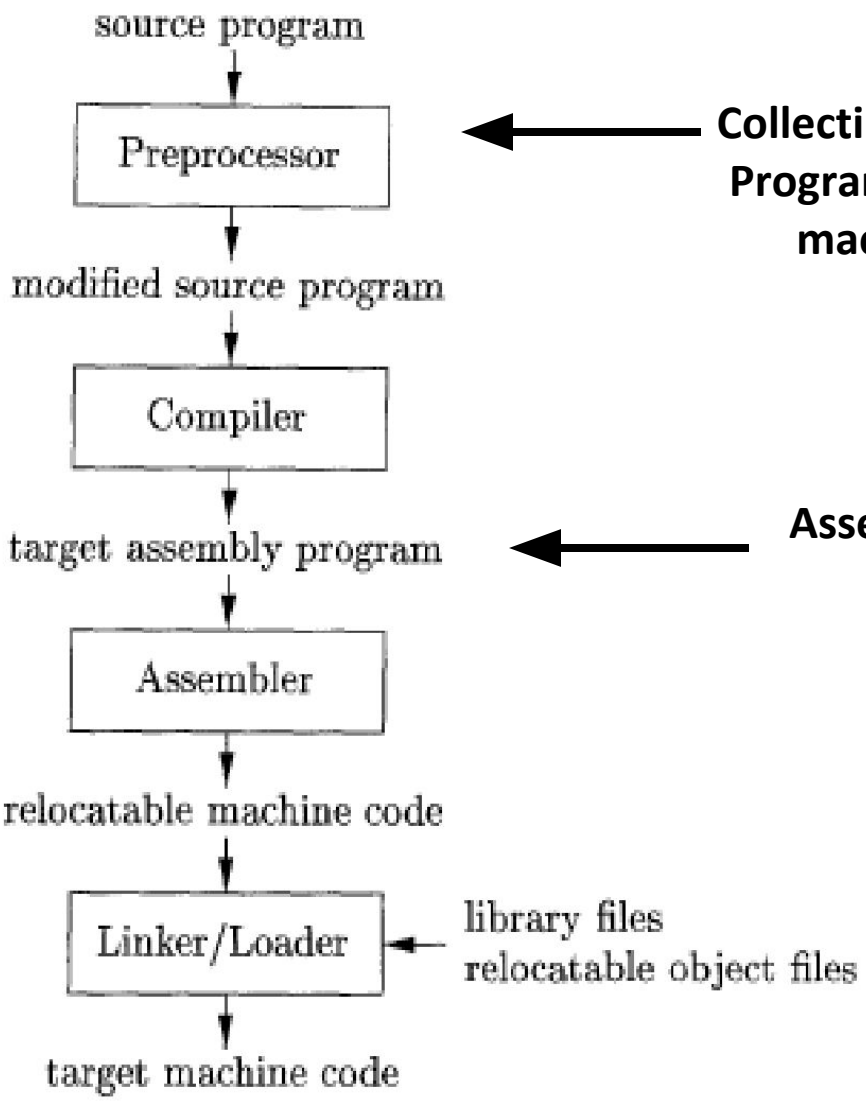
Principles, Techniques, & Tools

Plan

- Introduction
- Lexical analysis
- Syntax analysis
- Symbol tables

Language processing system

To create an executable target program several programs may be required



Collecting the source Program (modules, macros, etc.)

Assembly-language program

Introduction

-In order to reduce the complexity of designing and building computers, nearly all of these are made to execute relatively simple commands (but do so very quickly).

-A program for a computer must be built by combining these very simple commands into a program in what is *called machine language*.

-Most programming is done using a *high-level programming language* -> But this language can be very different from the machine language that the computer can execute.

===>This is where the compiler comes in

Introduction

- **Using a high-level language for programming has a large impact on how fast programs can be developed:**
  - Compared to machine language, the notation used by programming languages is ***closer*** to the way humans think about problems,
  - The compiler can spot some obvious ***programming mistakes***,
  - Programs written in a high-level language tend to be ***shorter*** than equivalent programs written in machine language
  - The ***same*** program can be compiled to ***many different*** machine languages and, hence, be brought ***to run on many*** different machines.

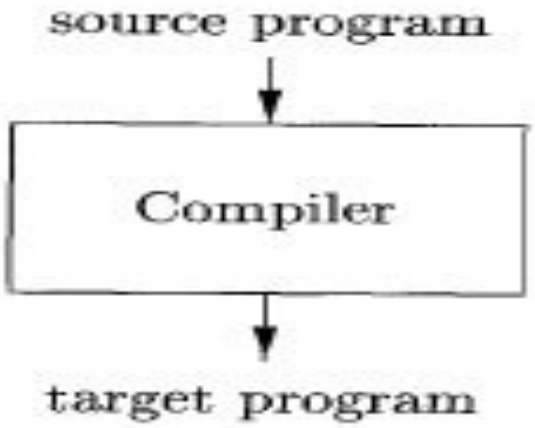
Introduction

What's a compiler ?

A compiler translates a program written in a high-level programming language that is suitable for human programmers into the low-level machine language that is required by computers

+

spots and reports obvious programmer mistakes.



Introduction

Running the target program

- **If the target program is an executable machine-language program, it can then be called by the user to process inputs and produce outputs.**

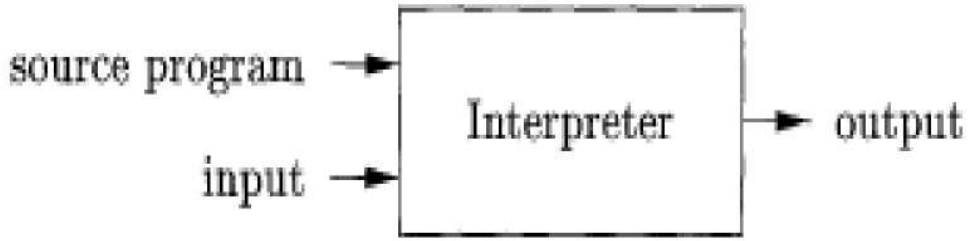
Introduction

What's an interpreter ?

- **An interpreter is another way of implementing a programming language.**
- **Interpretation shares many aspects with compiling (Lexing, parsing and type-checking)**

? But

Instead of producing a target program as a translation, an interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user



Compiler vs. Interpreter

- **An interpreter may need to process the same piece of the syntax tree (for example, the body of a loop) many times interpretation is slower than executing a compiled program.**

- **An interpreter executes the source program statement by statement it can usually give better error diagnostics than a compiler.**

Compilers

Principles, Techniques, & Tools

Plan

- Introduction
- Lexical analysis
- Syntax analysis
- Symbol tables

The Structure of a Compiler

Dividing into tokens (symbols : variable name, keyword or number)

Producing a tree-structure that reflects the structure of the program.

Determining if the program violates certain consistency requirements (if a variable is used but not declared, use a boolean value as a function pointer)

Symbol Table

Translating the intermediate language to assembly language (a textual representation of machine code) for a specific machine architecture.

Translating the assembly-language code into binary representation and addresses of variables, functions are determined.

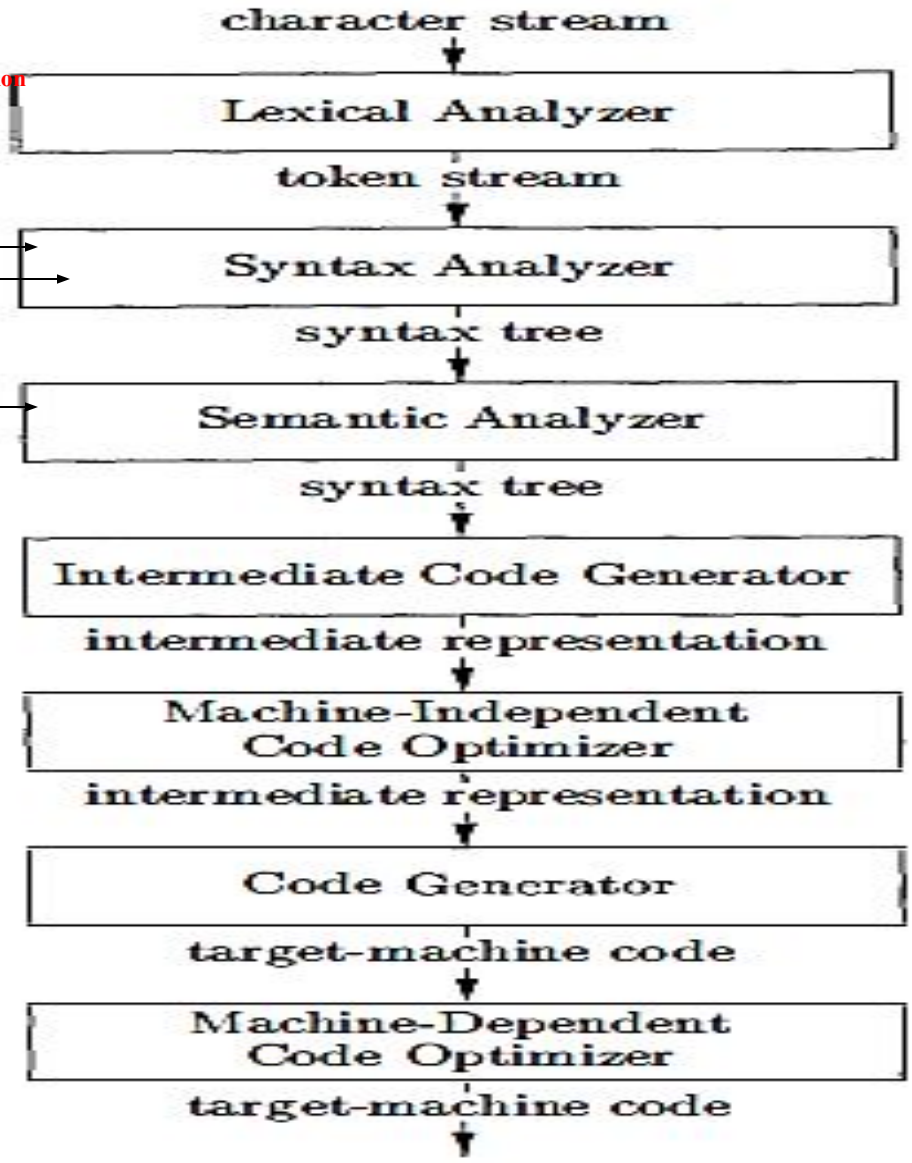


Figure 1.6: Phases of a compiler

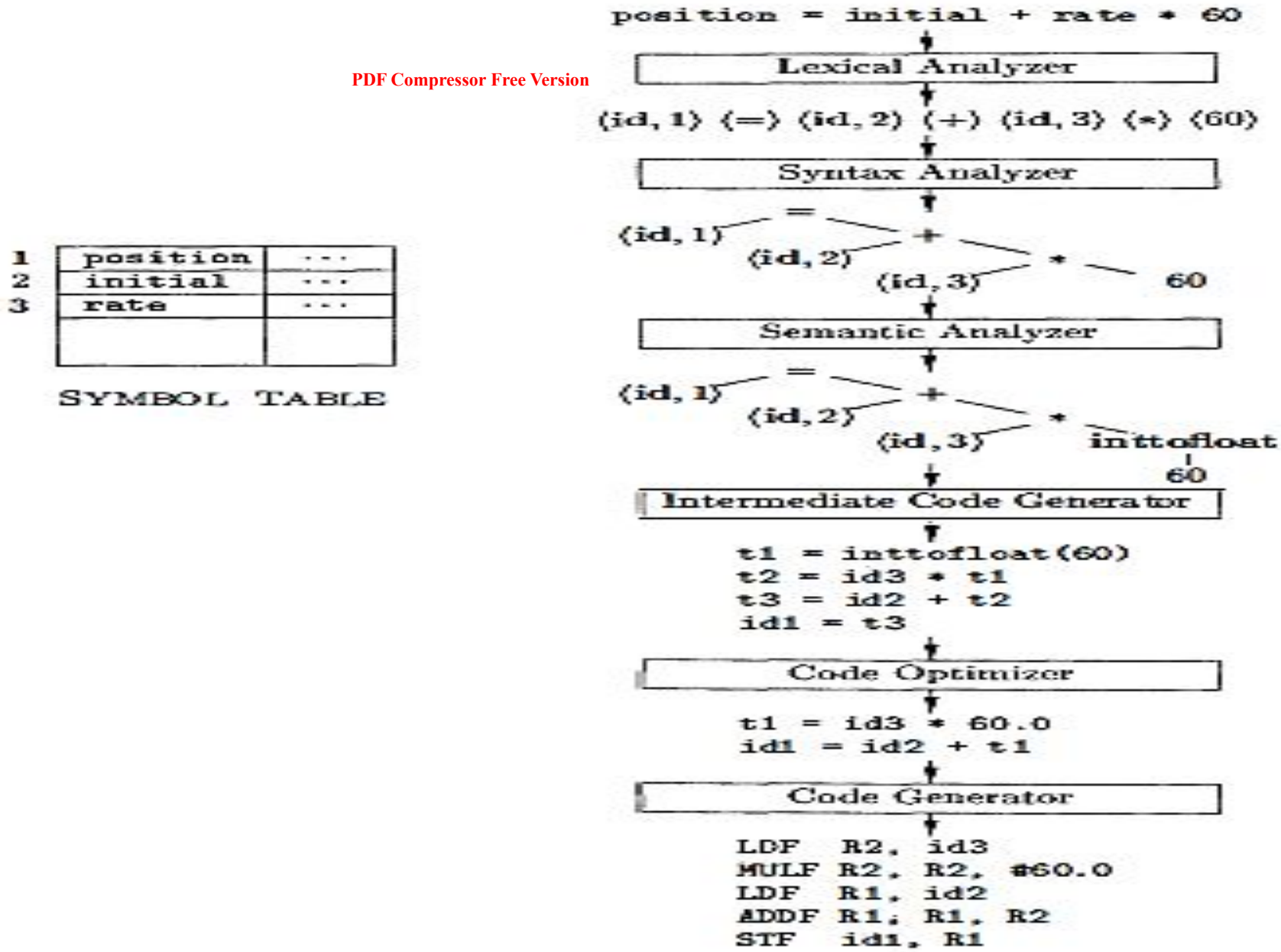


Figure 1.7: Translation of an assignment statement

Symbol Table Management

- The symbol table is a data structure containing a record for each variable name, with fields for the attributes of the name (storage allocated for a name, its type, its scope where in the program its value may be used), and for procedure names (number and types of its arguments, the method of passing each argument and the type returned).

- The data structure should be designed to allow the compiler to find the record for each name quickly and to store or retrieve data from that record quickly.

Introduction

Example

Overview of the Compiler

Lexical Analysis

Break input into *"TOKENS"*

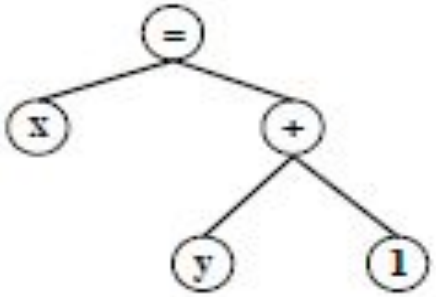
Source: `x = y + 1; /* incr x */ ...`

Tokens: ID, EQUALS, ID, PLUS, INT, SEMI, ...

Syntax Analysis

Context-Free Grammar

Build a parse tree



Semantic Analysis

Analyze types

Check for "semantic" errors

`x = y + true;`

Error

Introduction

Symbol Table

One entry for each identifier

key	type	address
w	bool	50
x	int	54
y	double	58
...	...	...

Introduction

Symbol Table

One entry for each identifier

key	type	address
w	bool	50
x	int	54
y	double	58
...	...	...

Intermediate Code

Not machine specific

```
temp1 := x
temp2 := temp1 + 1
x := temp2
```

Introduction

Symbol Table

One entry for each identifier

key	type	address
w	bool	50
x	int	54
y	double	58
...	...	...

Intermediate Code

Not machine specific

```
temp1 := x
temp2 := temp1 + 1
x := temp2
```

Code Optimization

Eliminate redundant data movement

Optimize "goto"s to other "goto" instructions

Introduction

Symbol Table

One entry for each identifier

key	type	address
w	bool	50
x	int	54
y	double	58
...	...	...

Intermediate Code

Not machine specific

```
temp1 := x
temp2 := temp1 + 1
x := temp2
```

Code Optimization

Eliminate redundant data movement

Optimize "goto"s to other "goto" instructions

Code Generation

Register Assignments

Machine Specific Code

```
mov.w     x,r5
add.w     #1,r5
mov.w     r5,x
```

Introduction

Symbol Table

One entry for each identifier

key	type	address
w	bool	50
x	int	54
y	double	58
...	...	...

Intermediate Code

Not machine specific

```
temp1 := x
temp2 := temp1 + 1
x := temp2
```

Code Optimization

Eliminate redundant data movement

Optimize "goto"s to other "goto" instructions

Code Generation

Register Assignments

Machine Specific Code

```
mov.w     x,r5
add.w     #1,r5
mov.w     r5,x
```

Error Handling

Can't just abort! ... Find more errors!

Patch things up and keep going

- Lexical Errors
- Syntactic Errors
- Semantic Errors