COMPILER CONSTRUCTION

Chapter 1

1. INTRODUCTION

What is a compiler?

- A computer program translates one language to another

- A compiler is a complex program
  - From 10,000 to 1,000,000 lines of codes
- Compilers are used in many forms of computing
  - Command interpreters, interface programs

What is the purpose of this text

- This text is to provide basic knowledge
  - Theoretical techniques, such as automata theory
- This text is to give necessary tools and practical experience
  - A series of simple examples
  - TINY, C-Minus

Why Compiler

- Writing machine language-numeric codes is time consuming and tedious

C7 06 0000 0002

Mov x, 2

X=2

- The assembly language has a number of defects
  - Not easy to write
  - Difficult to read and understand

1.2 Programs related to Compiler

Interpreters

- Execute the source program immediately rather than generating object code
- Examples: BASIC, LISP, used often in educational or development situations
- Speed of execution is slower than compiled code by a factor of 10 or more
- Share many of their operations with compilers

Assemblers

- A translator for the assembly language of a particular computer
- Assembly language is a symbolic form of one machine language
- A compiler may generate assembly language as its target language and an assembler finished the translation into object code

Linkers

- Collect separate object files into a directly executable file
- Connect an object program to the code for standard library functions and to resource supplied by OS
- Becoming one of the principle activities of a compiler, depends on OS and processor

Loaders

- Resolve all re-locatable address relative to a given base
- Make executable code more flexible
- Often as part of the operating environment, rarely as an actual separate program

Preprocessors

- Delete comments, include other files, and perform macro substitutions
- Required by a language (as in C) or can be later add-ons that provide additional facilities

Editors

- Compiler have been bundled together with editor and other programs into an interactive development environment (IDE)
- Oriented toward the format or structure of the programming language, called structure-based
- May include some operations of a compiler, informing some errors

Debuggers

- Used to determine execution error in a compiled program
- Keep tracks of most or all of the source code information
- Halt execution at pre-specified locations called breakpoints
- Must be supplied with appropriate symbolic information by the compiler

Profiles

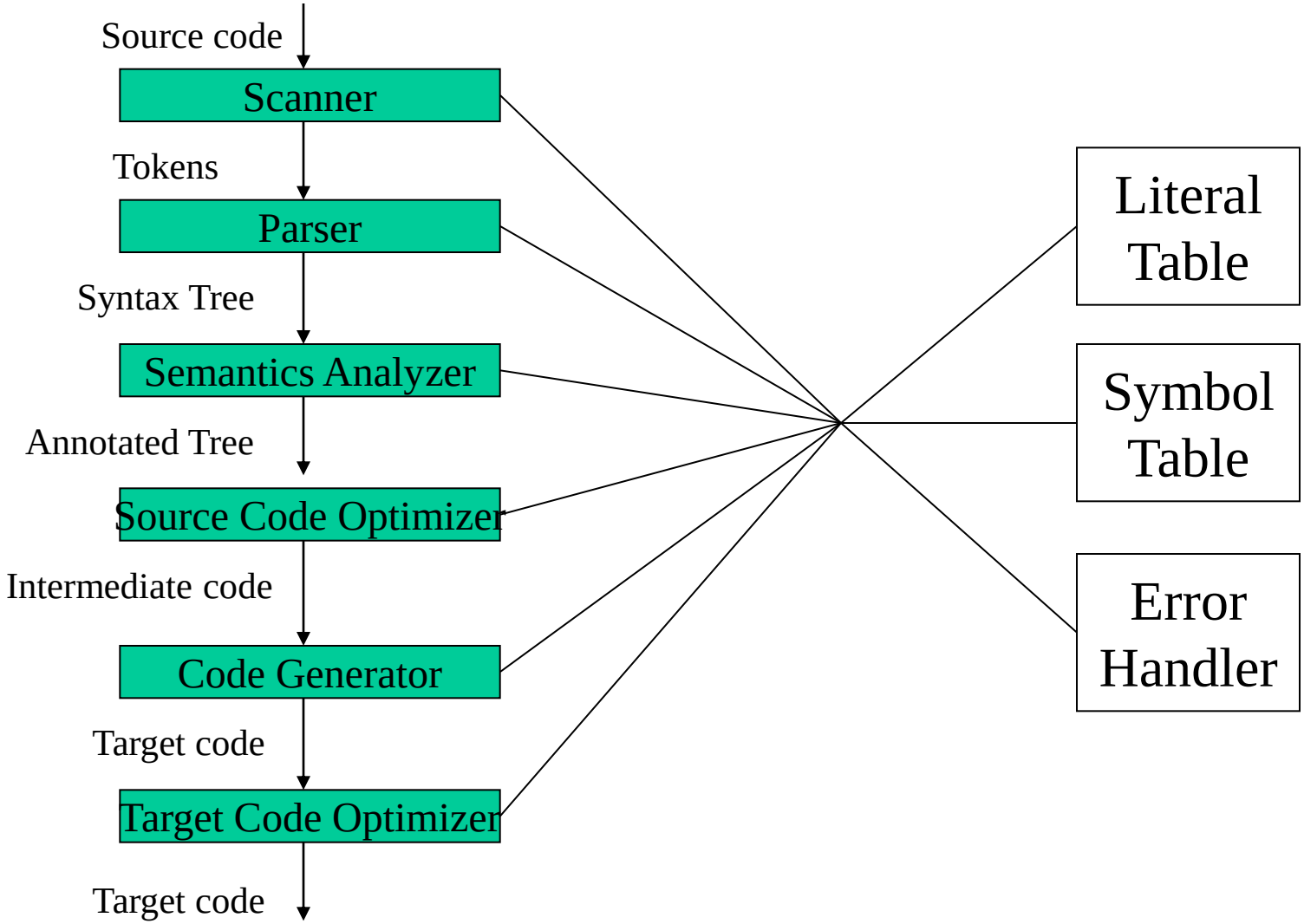
- Collect statistics on the behavior of an object program during execution
  - Called Times for each procedures
  - Percentage of execution time
- Used to improve the execution speed of the program

1.3 The Translation Process

The phases of a compiler

- Six phases
  - Scanner
  - Parser
  - Semantic Analyzer
  - Source code optimizer
  - Code generator
  - Target Code Optimizer
- Three auxiliary components
  - Literal table
  - Symbol table
  - Error Handler

The Phases of a Compiler



The Scanner

- Lexical analysis: it collects sequences of characters into meaningful units called tokens
- An example: a[index]=4+2
  - a identifier
  - [left bracket
  - index identifier
  -] right bracket
  - = assignment
  - 4 number
  - + plus sign
  - 2 number
- Other operations: it may enter literals into the literal table

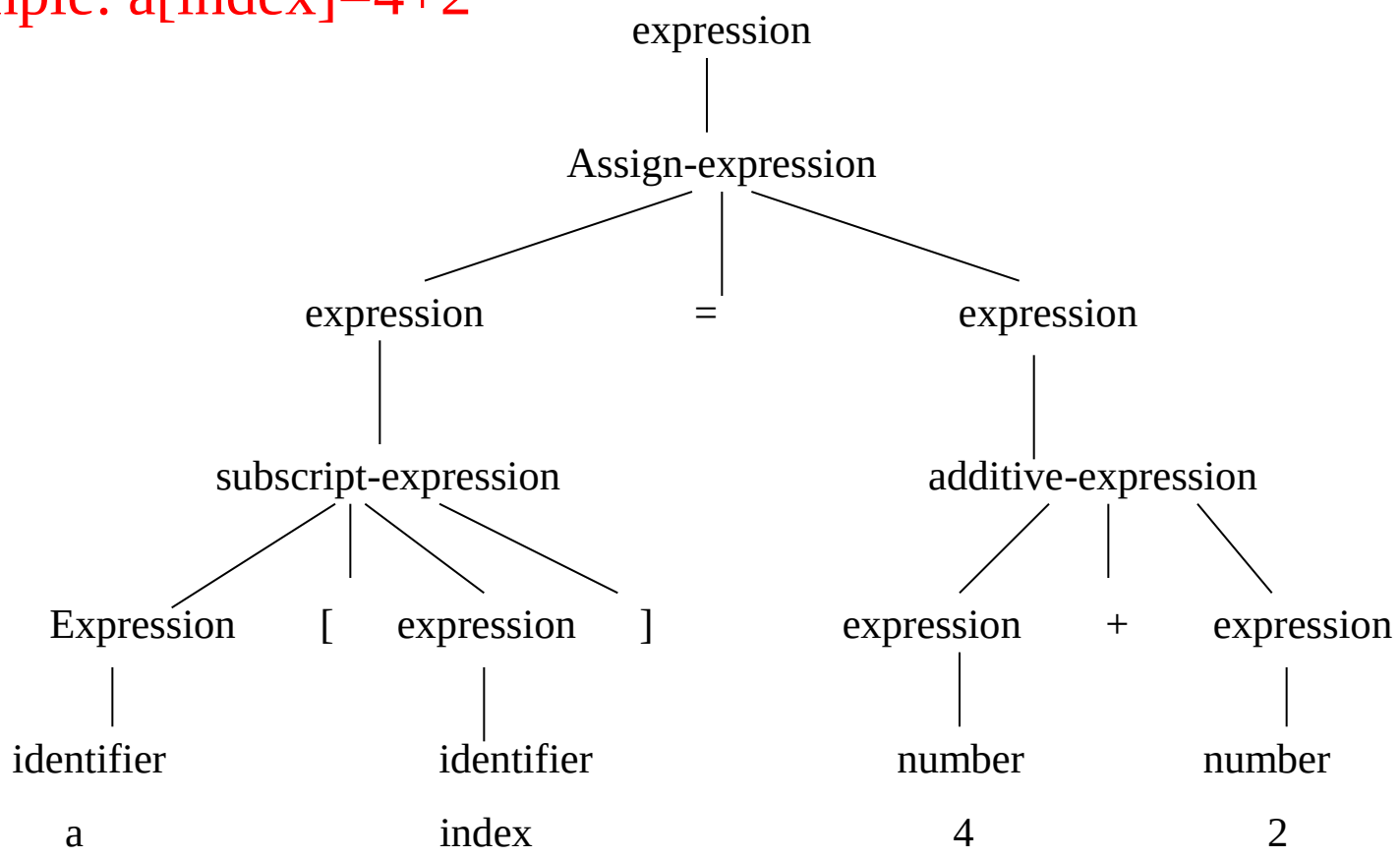
RETURN

The Parser

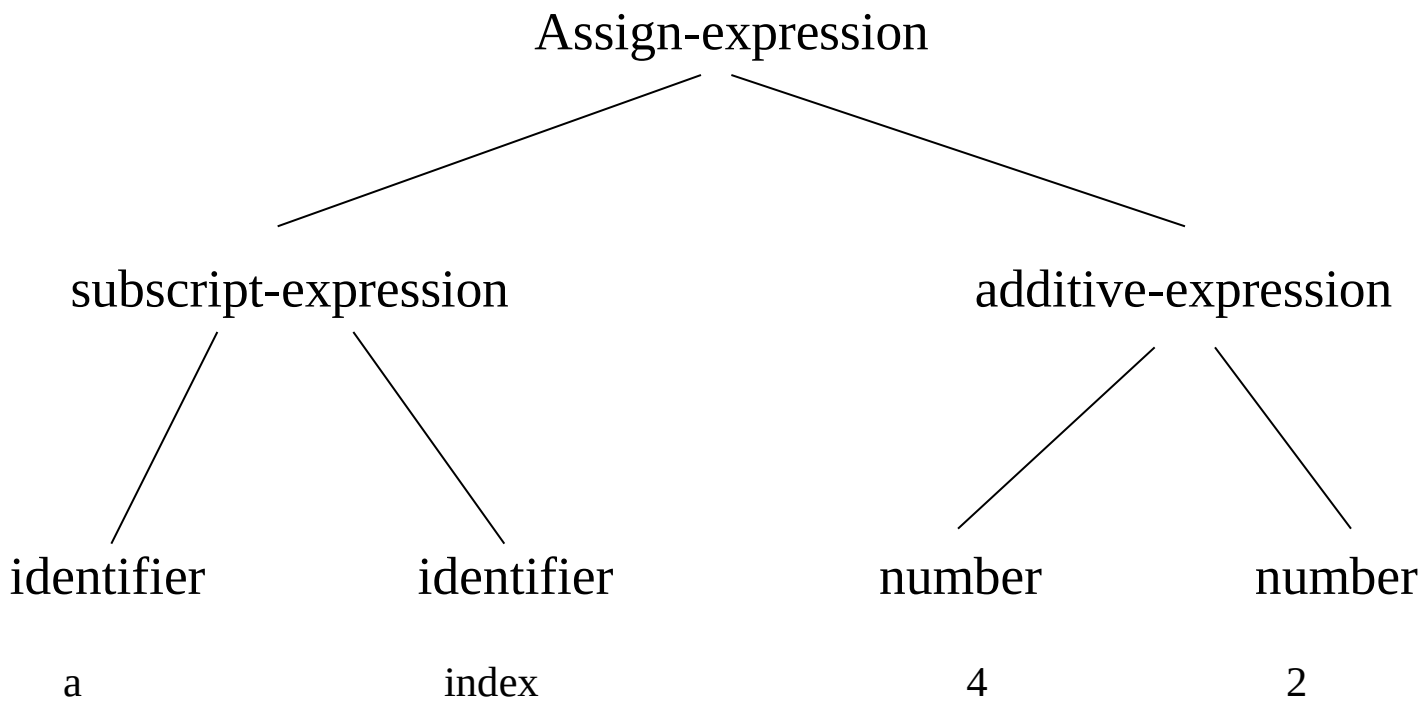
- Syntax analysis: it determines the structure of the program
- The results of syntax analysis are a parse tree or a syntax tree
- An example: a[index]=4+2
  - Parse tree
  - Syntax tree (abstract syntax tree)

The Parse Tree

An example: a[index]=4+2



The Syntax Tree

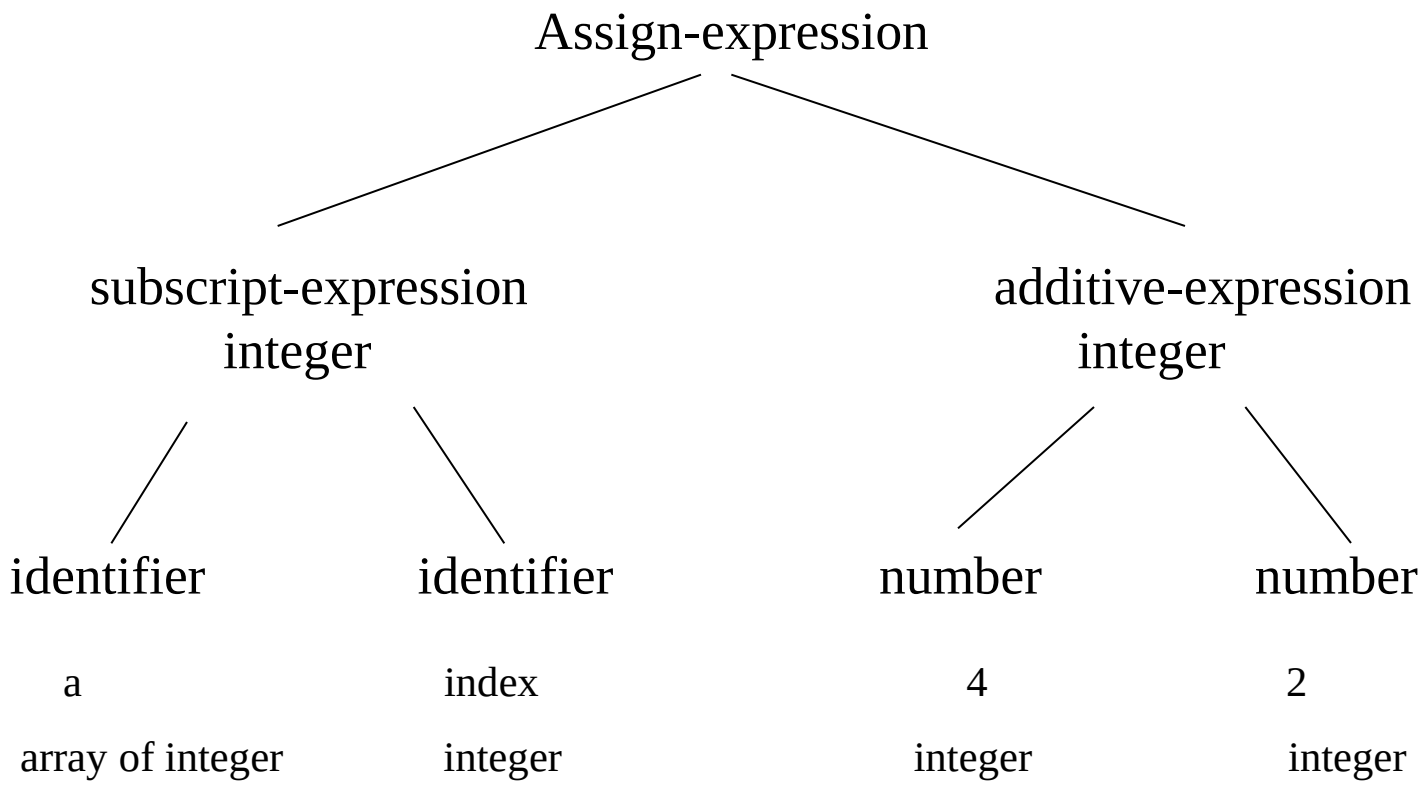


RETURN

The Semantic Analyzer

- The semantics of a program are its “meaning”, as opposed to its syntax, or structure, that
  - determines some of its running time behaviors prior to execution.
- Static semantics: declarations and type checking
- Attributes: The extra pieces of information computed by semantic analyzer
- An example: a[index]=4=2
  - The syntax tree annotated with attributes

The Annotated Syntax Tree

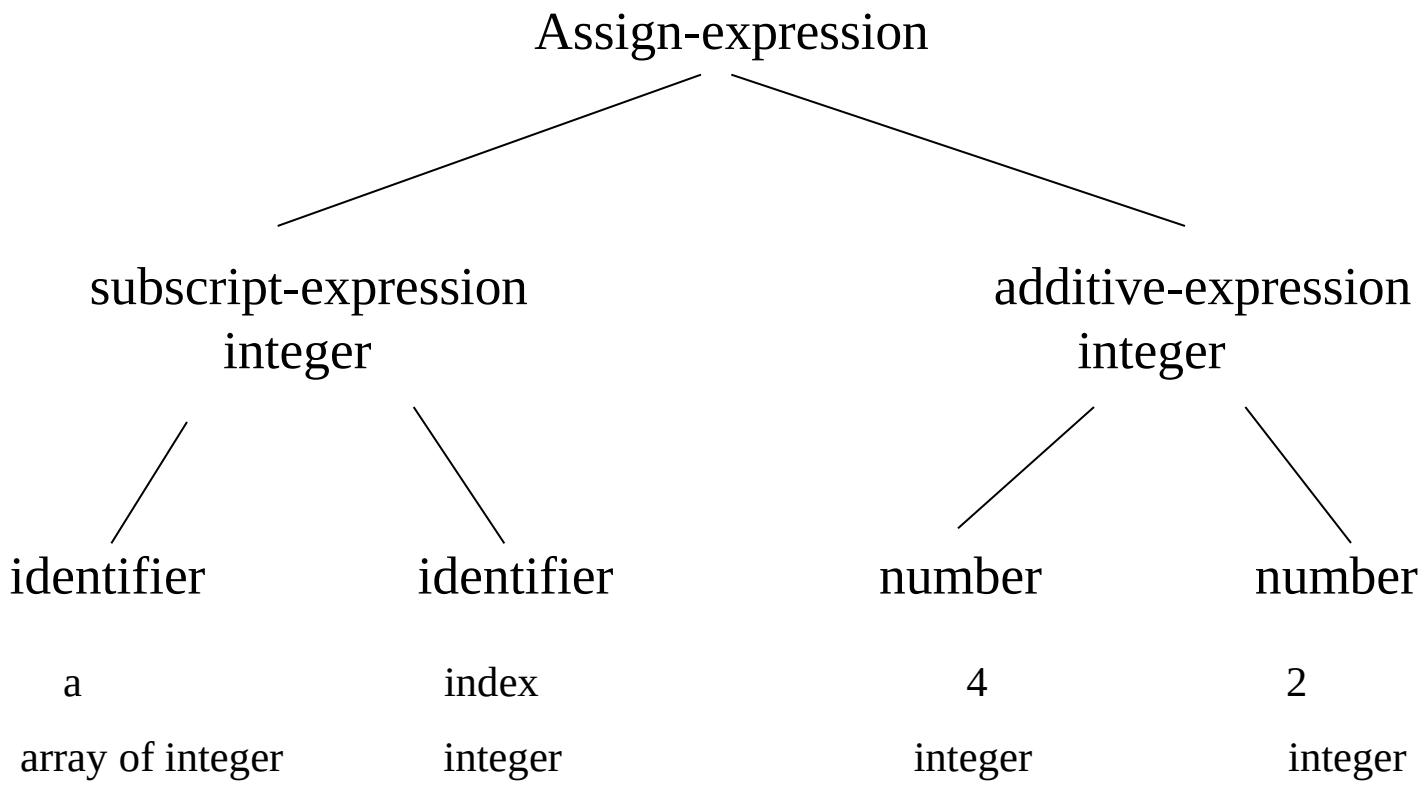


RETURN

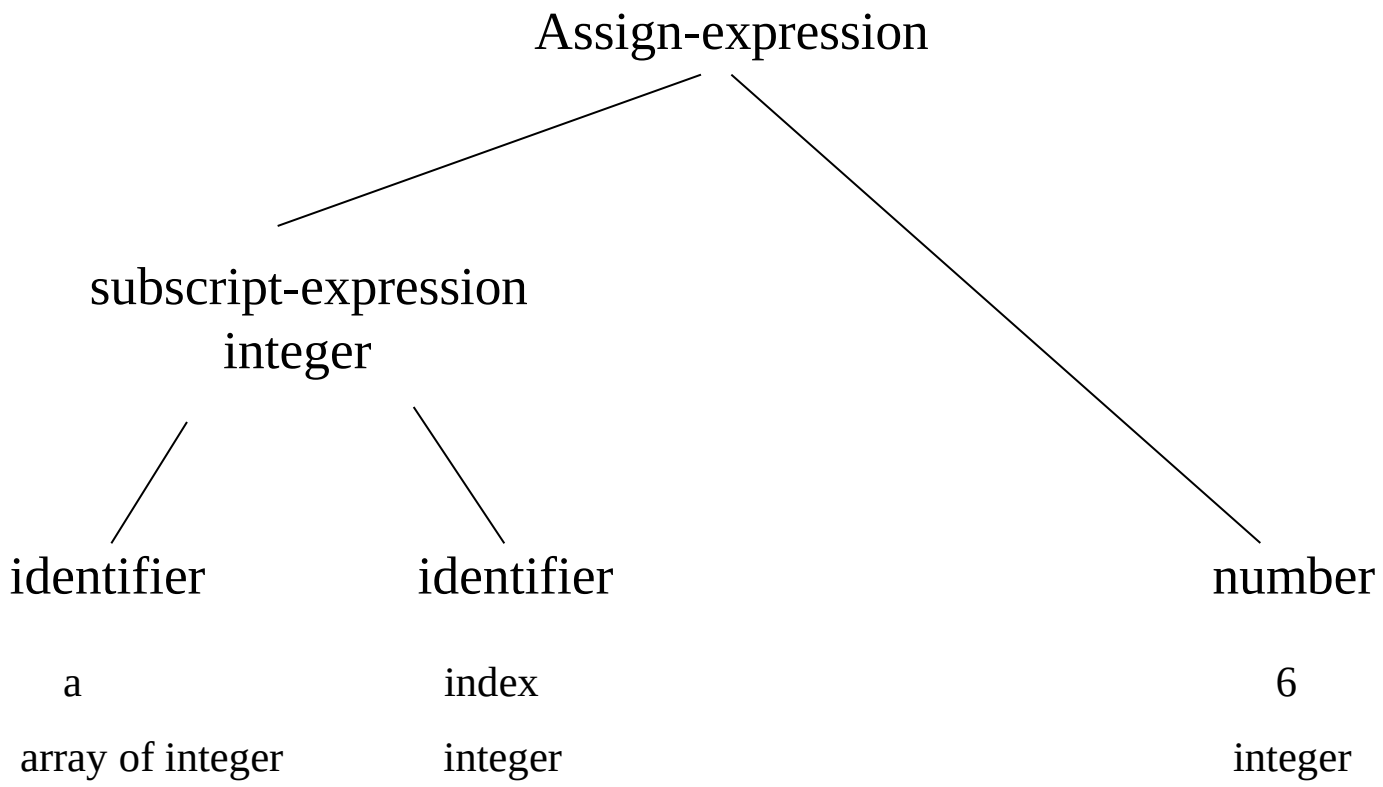
The Source Code Optimizer

- The earliest point of most optimization steps is just after semantic analysis
- The code improvement depends only on the source code, and as a separate phase
- Individual compilers exhibit a wide variation in optimization kinds as well as placement
- An example: a[index]=4+2
  - Constant folding performed directly on annotated tree
  - Using intermediate code: three-address code, p-code

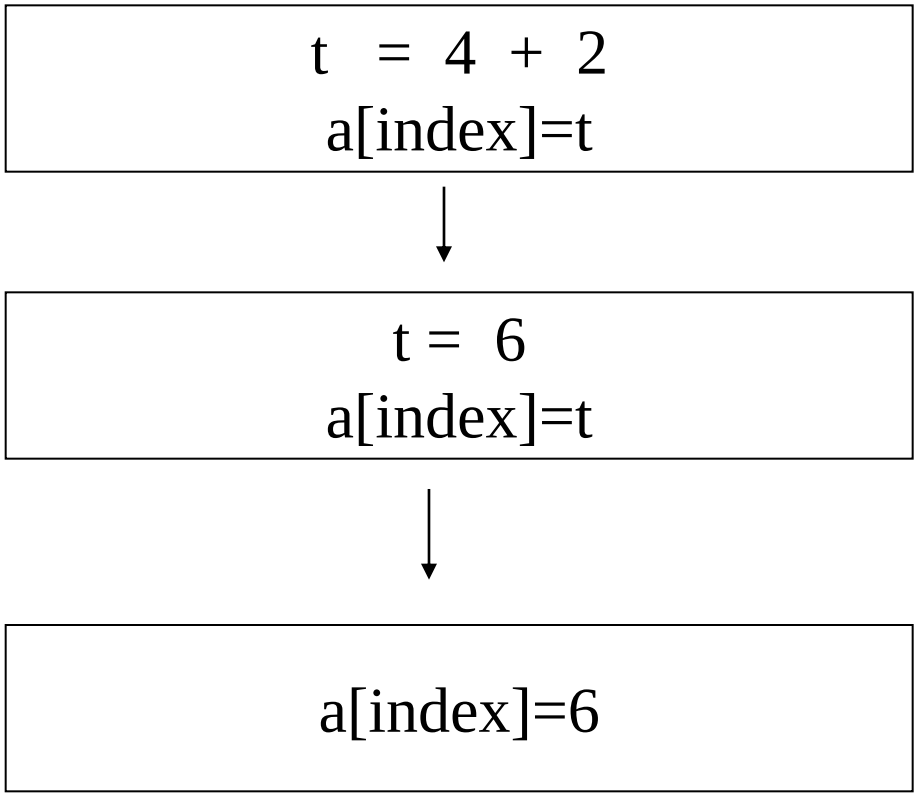
Optimizations on Annotated Tree



Optimizations on Annotated Tree



Optimization on Intermediate Code

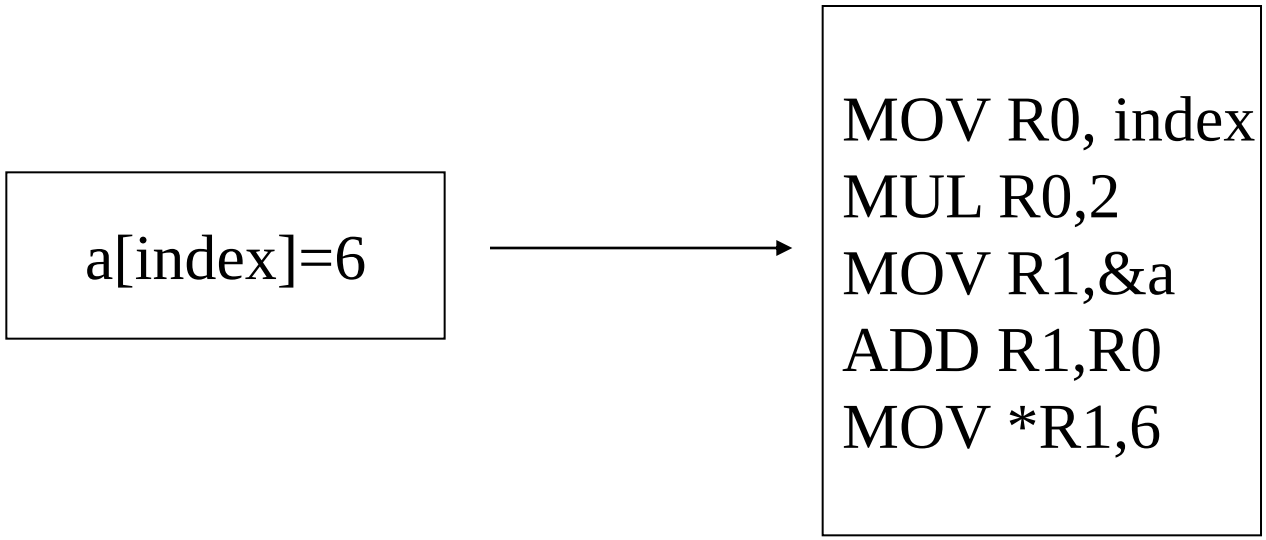


RETURN

The Code Generate

- It takes the intermediate code or IR and generates code for target machine
- The properties of the target machine become the major factor:
  - Using instructions and representation of data
- An example: a[index]=4+2
  - Code sequence in a hypothetical assembly language

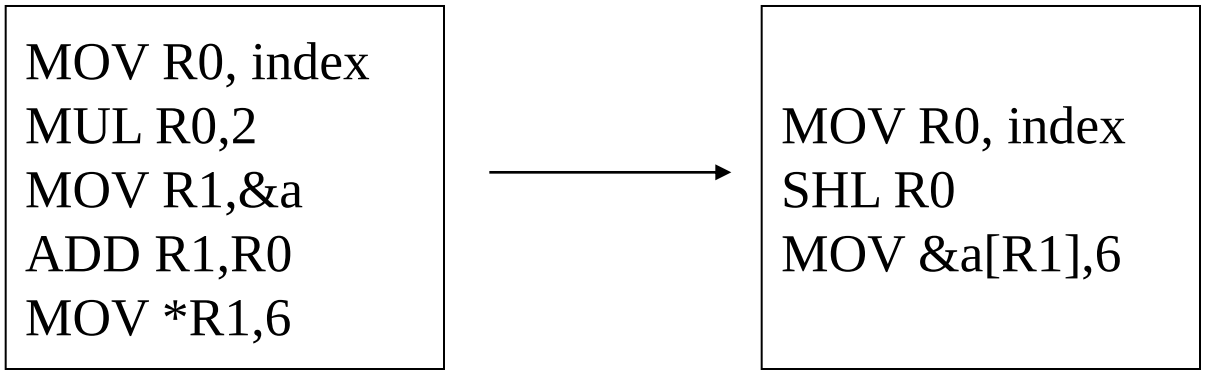
A possible code sequence



RETURN

The Target Code Optimizer

- It improves the target code generated by the code generator:
  - Address modes choosing
  - Instructions replacing
  - As well as redundant eliminating



BACK

1.4 Major Data Structure in a Compiler

Principle Data Structure for Communication among Phases

- TOKENS
  - A scanner collects characters into a token, as a value of an enumerated data type for tokens
  - May also preserve the string of characters or other derived information, such as name of identifier, value of a number token
  - A single global variable or an array of tokens
- THE SYNTAX TREE
  - A standard pointer-based structure generated by parser
  - Each node represents information collect by parser or later, which maybe dynamically allocated or stored in symbol table
  - The node requires different attributes depending on kind of language structure, which may be represented as variable record.

Principle Data Structure for Communication among Phases

- THE SYMBOL TABLE
  - Keeps information associated with identifiers: function, variable, constants, and data types
  - Interacts with almost every phase of compiler.
  - Access operation need to be constant-time
  - One or several hash tables are often used,
- THE LITERAL TABLE
  - Stores constants and strings, reducing size of program
  - Quick insertion and lookup are essential