Compiler Construction

Introduction

Department of Computer Science

King Saud University

Thanks for Dr. Mohsen DENGUIR for his effort in producing the ppt

What is a compiler?

- A compiler is a language translator that takes as input a program written in a high level language and produces an equivalent program in a low-level language.
- For example, a compiler may translate a C program into an executable program running on a SPARC processor.
- In the process of translation, a compiler goes through several phases:
  - Lexical analysis (also called scanning)
  - Syntax analysis (also called parsing)
  - Semantic analysis
  - Optimization (not in this course!)
  - Code generation

Lexical Analysis

- The job of the *lexical analyzer*, or *scanner*, is to read the source program one character at a time and produce as output a stream of tokens (we discuss these next)
- The tokens produced by the scanner serve as input to the next phase, the parser.
- Thus, the lexical analyzer's job is to translate the source program into a form more conducive to recognition by the parser.

Tokens

- Tokens are used to represent low-level program units such as
  - .Identifiers, such as sum, value, and x
  - .Numeric literals, such as 123 and 1.35e06
  - .Operators, such as +, *, &&, <=, and %
  - .Keywords, such as if, else, and return
  - Many other language symbols

Classes of Tokens

- There are many ways we could represent the tokens of a programming language. One possibility is to use a 2-tuple of the form <token_class, value>
- For example, consider the token class identifier. The identifiers sum and value may be represented as <ident, "sum"> and <ident, "value">, respectively.
- The token class NumericLiteral may be represented in the same way; for example, the literals 123 and 1.35e06 may be represented as <NumericLiteral, "123"> and <NumericLiteral, "1.35e06">, respectively.
- The same applies to operators; for example, <relop, ">="> and <addop, "-">

Representing Tokens

- These 2-tuples are easily represented as a struct in C:

```
typedef enum _TokenClass {ident, numlit,...} TokenClass;
struct Token {
        TokenClass tokenClass;
        char *tokenValue;
};
```

Tokens: an Example

The scanner may take the expression

```
x = 2 + f(3);
```

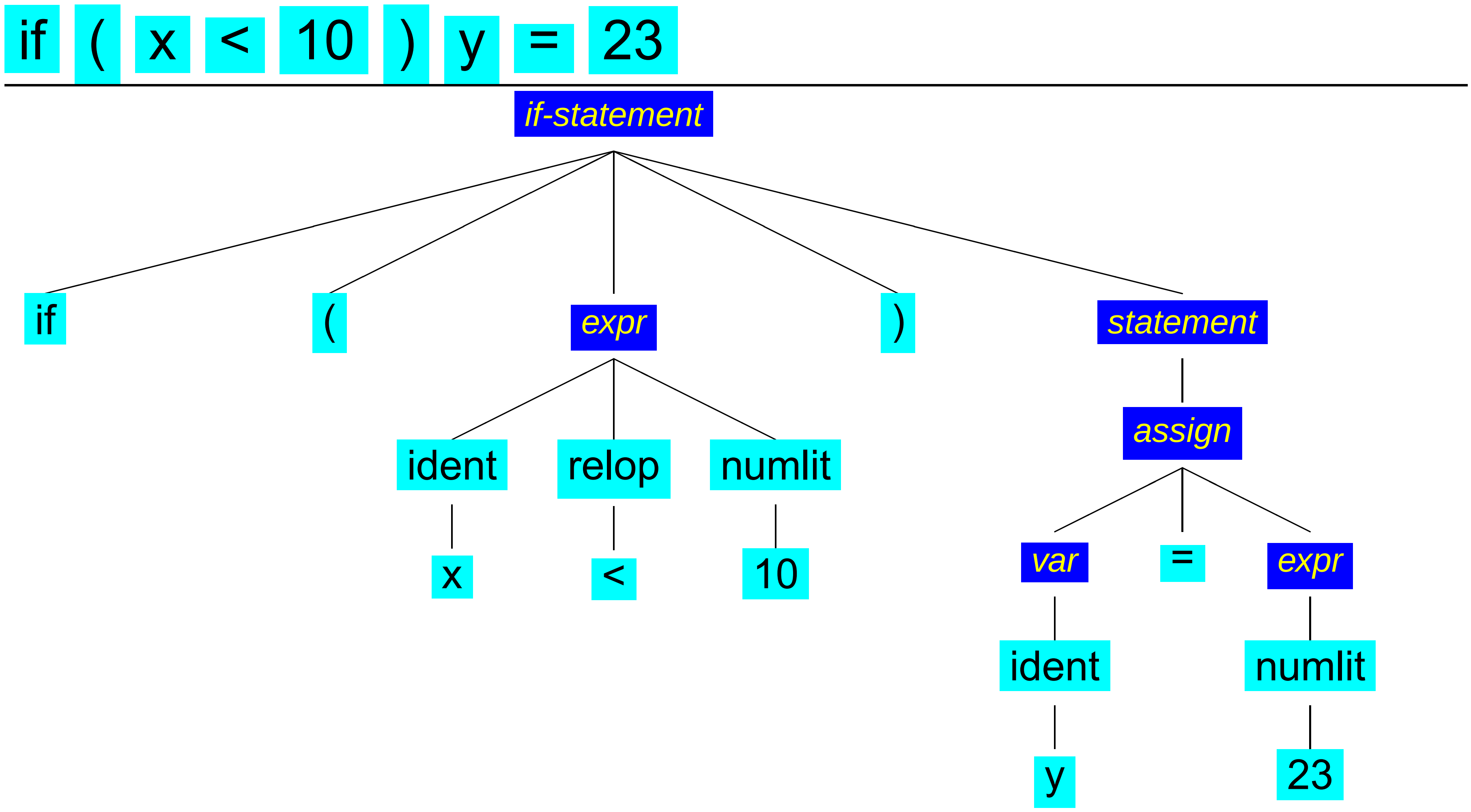
and produce the following stream of tokens

```
<ident, "x">
<assign_op, "=">
<numlit, "2">
<addop, "+">
<ident, "f">
<lparen, "(">
<numlit, "3">
<rparen, ")">
<semicolon, ";">
```

Syntax Analysis

- The job of the *syntax analyzer*, or *parser*, is to take a stream of tokens produced by the lexical analyzer and build a parse tree (or syntax tree).
- The parser is basically a program that determines if sentences in a language are constructed properly according to the rules of the language.

A Parse Tree


if (x < 10) y = 23
if-statement
if
(
expr
)
statement
ident
relop
numlit
assign
x
<
10
var
=
expr
ident
numlit
y
23

Syntax Analysis

- There are two general categories of parsers:
    - Top down parsers, which include
        - LL(1) table-driven parsers
        - Recursive descent parsers (we will write one!)
    - Bottom up table driven parsers (table-driven)
        - SLR (simple LR)
        - LR(1) parsers
        - LALR(1) parsers
- The syntax of a language is defined by using a context free grammar (CFG).
- A CFG uses BNF rules to describe the syntax:

  IfStatement ⟶ **’if’** **’(’** *Expr* **’)’** *Statement [* **’else’** *Statement]*

Semantic Analyzer

- The semantic analyzer's job is to attach some meaning to the structure produced by the parser.
- Activities include:
  - Ensuring an identifier is defined before being used in a statement or expression.
  - Enforcing the scope rules of the language.
  - Performing type checking
  - Producing intermediate code

Semantic Analysis

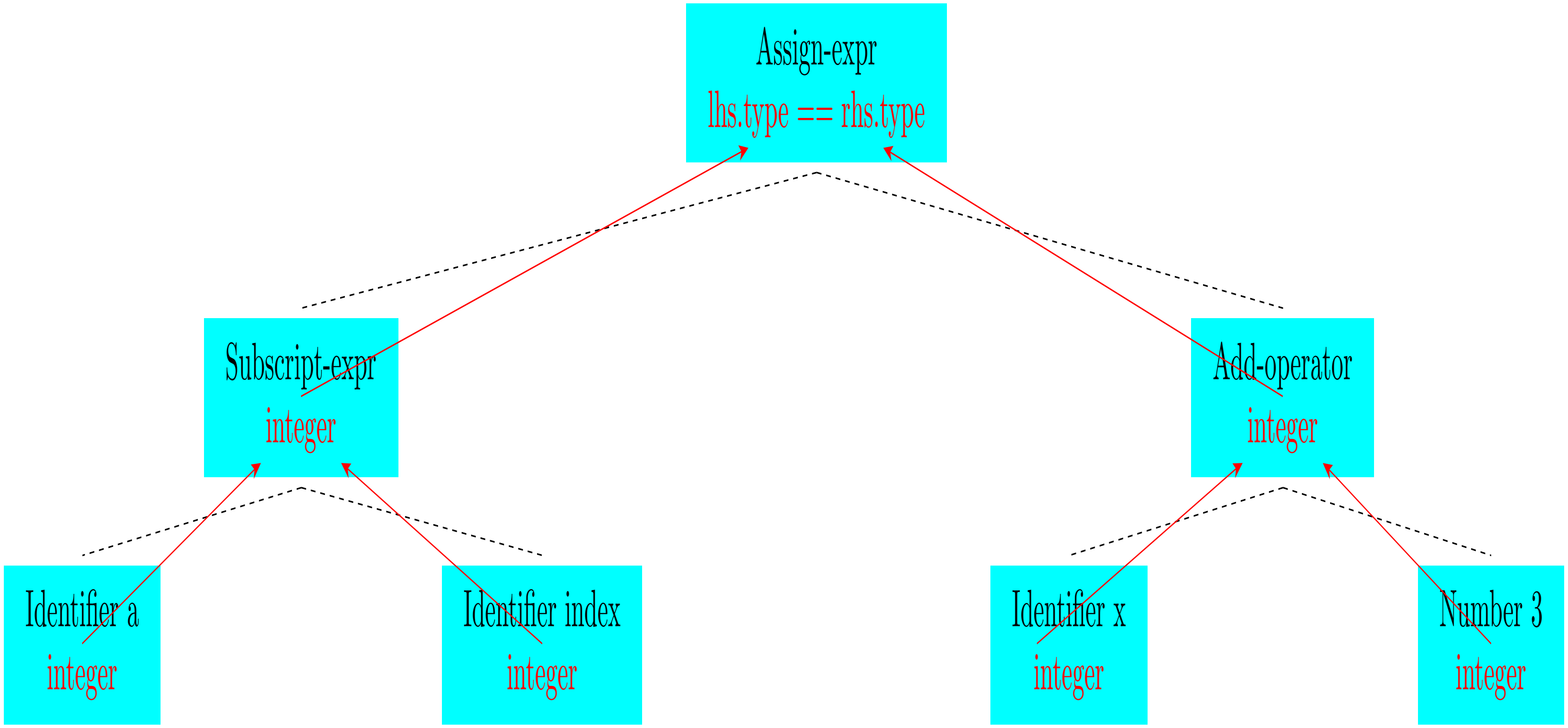
- .Static semantics can be determined by the compiler prior to execution, including
  - Declarations
    - Determine the structure and attributes of a user-defined data type
    - Determine type of a variable
    - Determine the number and types of parameters of a procedure
  - Type checking
    - The process of ensuring that the type(s) of the operand(s) are appropriate for an operation

Semantic Analysis

- .Attributes are extra pieces of information computed by the semantic analyzer. These include the types of variables, constants, operators, etc.
- An annotated syntax tree is a syntax tree that has been "decorated" with attributes.
  - *Inherited attributes* come down the syntax tree from parent or sibling nodes
  - *Synthesized attributes* come up the syntax tree from child nodes

Semantic Analysis: an Example

Annotated syntax tree: a[index] = x + 3



Semantic Analysis

- Some optimization may be done during this phase:
  - Source code optimization (e.g., constant folding):
    - .X := 2 + 4; can be optimized to X := 6;
  - Intermediate code optimization:
    - .Temp := 5; A[index] := Temp can be optimized to A[index] := 5;