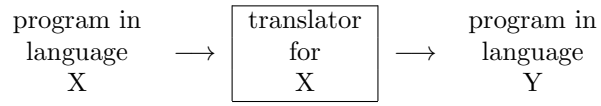
CPS 140 - Mathematical Foundations of CS
Dr. S. Rodger
Section: The Structure of a Compiler

1.1 What is a Compiler?

I. Translator

Definition:



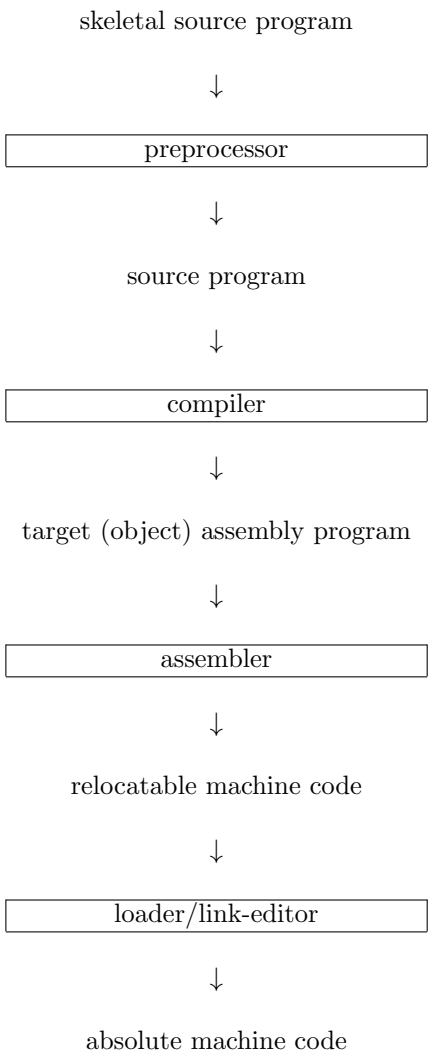
Examples:

Source Language	Object Language	Name	Example
High Level	High Level	preprocessor	ratfor → f77 m4, cpp
Assembly	Machine	assembler	as
High Level	Machine	compiler	g++, javac
Any	executes immediately	interpretor	BASIC (often) c shell apl, lisp, java

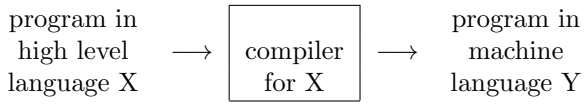
- Preprocessor

```
for i=1 to n do
    (stmts)
end for

    ↓

i = 1
while (i<=n) do
    (stmts)
    i = i + 1
end while
```

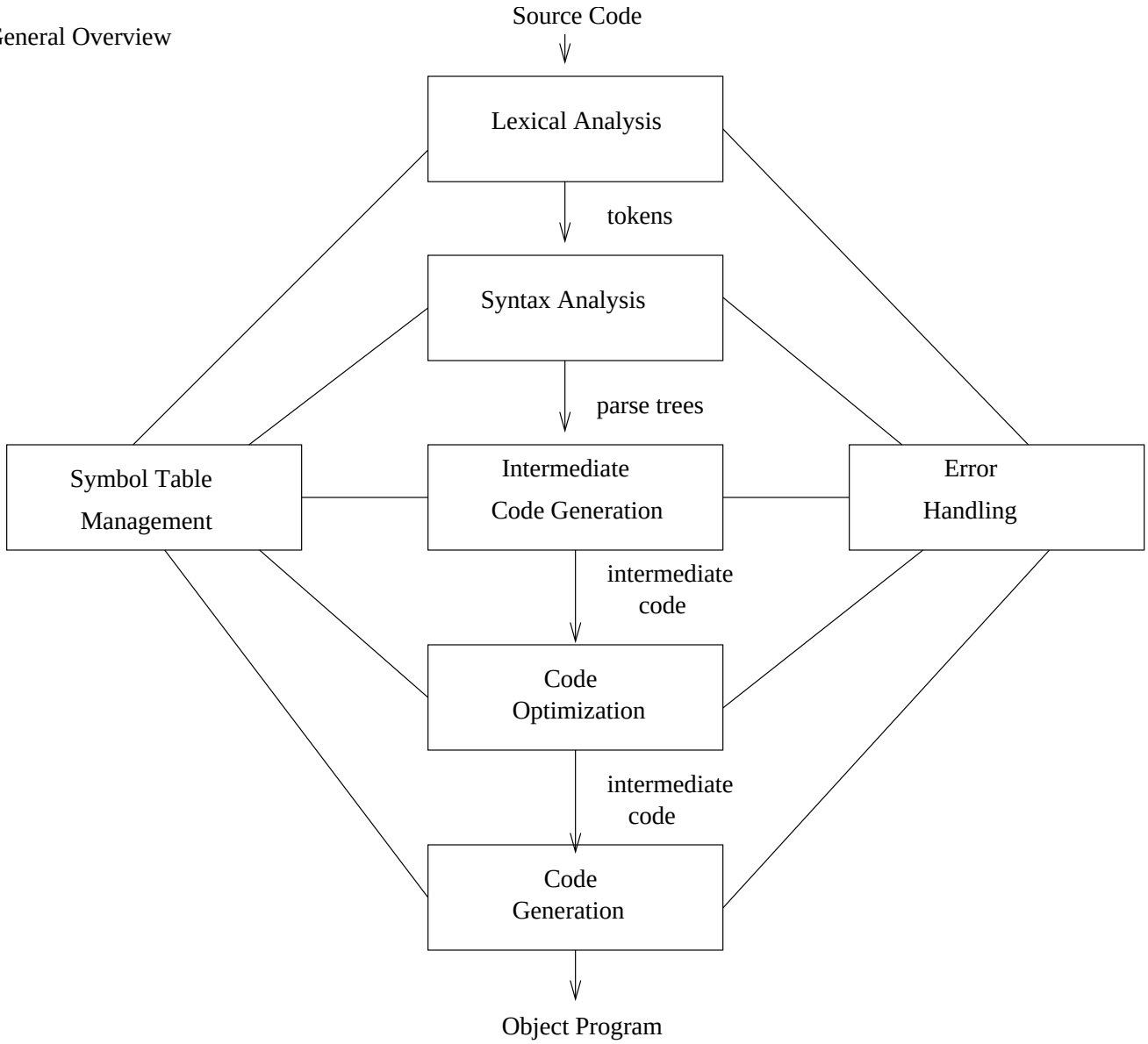
II. Language Processing System



III. Compiler



General Overview



1.3 PHASES OF COMPILATION

1.3.1 Lexical Analysis (Scanner)

a. Purpose: Read the same program character by character grouping them into atomic units called "tokens."

b. Tokens:

- depend on language and compiler writer
- Examples:

reserved words	if, for
operators	$+, -, <, =$
constants	0, 4.89
punctuation	(, }, [
identifiers	sb, ch

- treated as a pair: token.type and token.value
  - token type is a (mnemonic) integer
  - some tokens have no value

c. Example

if (x <= 0) x = y + z

when put through lexical analyzer produces:

token	type	value
if	25	
(	28	
id	23	"x"
<=	27	
int constant	22	0
)	38	
id	23	"x"
= assgnment	4	
id	23	"y"
+	34	
id	23	"z"

d. How does one build a lexical analyzer?

- from scratch
- lex

e. Preview of Lex

- idea: tokens described by regular expressions
- basic syntax:
  regular expression, action
- basic semantics:
  if match regular expression, then do action.
- Example:

```
%%
"if"      return(25);
"("       return(28);
[0-9]+    return(22);
```

f. Remarks

Besides returning token types and values, the lexical analyzer might

a) print error messages

b) insert identifiers in the symbol table

1.3.2 Syntax Analysis (Parsing)

a. Purpose: Accepts the sequence of tokens generated by the lexical analyzer, checks whether the program is syntactically correct, and generates a parse tree.

b. Syntax: formally described by a context free grammar.

c. Parse Tree

if (x<=0) x = y + z

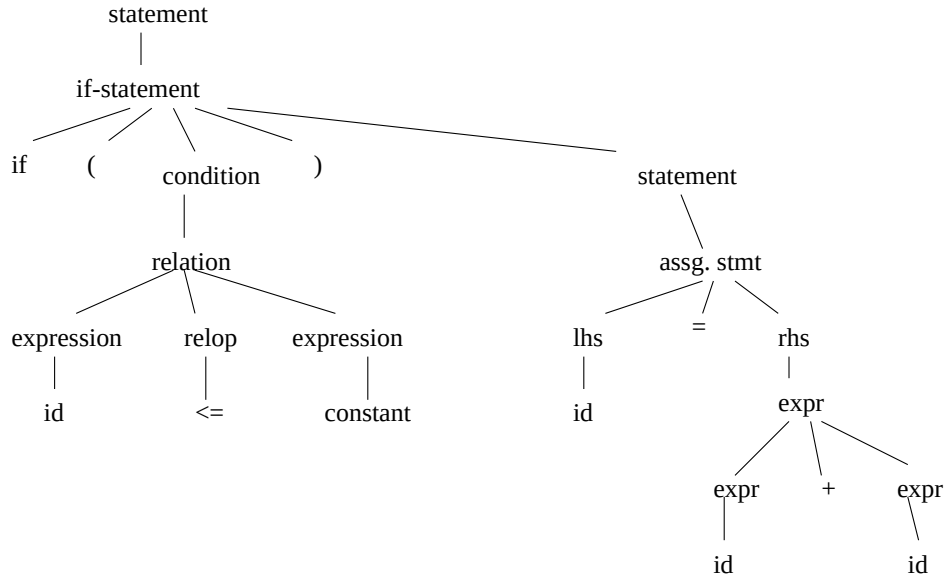


Figure 2 is the parse tree for this statement.

d. How does one build a parser?

- from scratch
- using a parser generator such as yacc

1.3.3 Intermediate Code Generator

a. Purpose: Traverse the parse tree, producing simple intermediate code.

b. Three-Address Code:

Instructions:

1. id := id op id
2. goto label
3. if condition goto label

Example:

```
if    (x<=0) x = x + z

            ↓

      if (x<=0) goto L1
      goto L2
L1:   x := y + z
L2:
```

1.3.4 Intermediate Code Generation

a. Purpose: Transform the intermediate code into "better" code.

b. Examples

1) Rearrangement of Code

```
      if (x<=0) goto L1                  if (x>0 goto L2
      goto L2              →             x = y + z
L1:   x = y + z                  L2:
L2:
```

2) Redundancy Elimination

```
a = w + x + y                            T1 = x + y
                           →             a = w + T1
b = x + y + z                            b = T1 + z
```

3) Strength Reduction

```
x^2          →          x * x
expensive →             cheap
operator                operator
```

4) Frequency Reduction

```
for (i=1; i<n; i=i+1) {          T1 = sqrt(26)
      x = sqrt(26)        →      for (i=1; i<n; i=i+1) {
      }                                x = T1
                                       }
```

c. Remarks:

1) Main criteria for optimization is speed.

1.3.5 Code Generation

a. Purpose: Transform intermediate code to machine code (assembler)

b. Example: a = b + c

```
mov   b, R1
add   c, R1
mov   R1, a
```

c. Remarks

1) completely machine dependent whereas other phases are not

2) "register allocation" is the most difficult task

- idea - use registers (fast access) to avoid memory use (slow access)
- problem - only a finite number of registers (during intermediate code phase, one assumes an infinite number)

1.4 Symbol Table

a. Purpose: record information about various objects in the source program

b. Examples

- procedure - no. and type of arguments
- simple variable - type
- array - type, size

c. Use - information is required during

- parsing
- code generation

1.5 Error Handler

a. Errors - all errors should be

- detected
- detected correctly
- detected as soon as possible
- reported at the appropriate place and in a helpful manner

b. Purpose

- report errors
- "error recovery" - proceed with processing

c. Note: Errors can occur in each phase

- misspelled token
- wrong syntax
- improper procedure call
- statements that cannot be reached