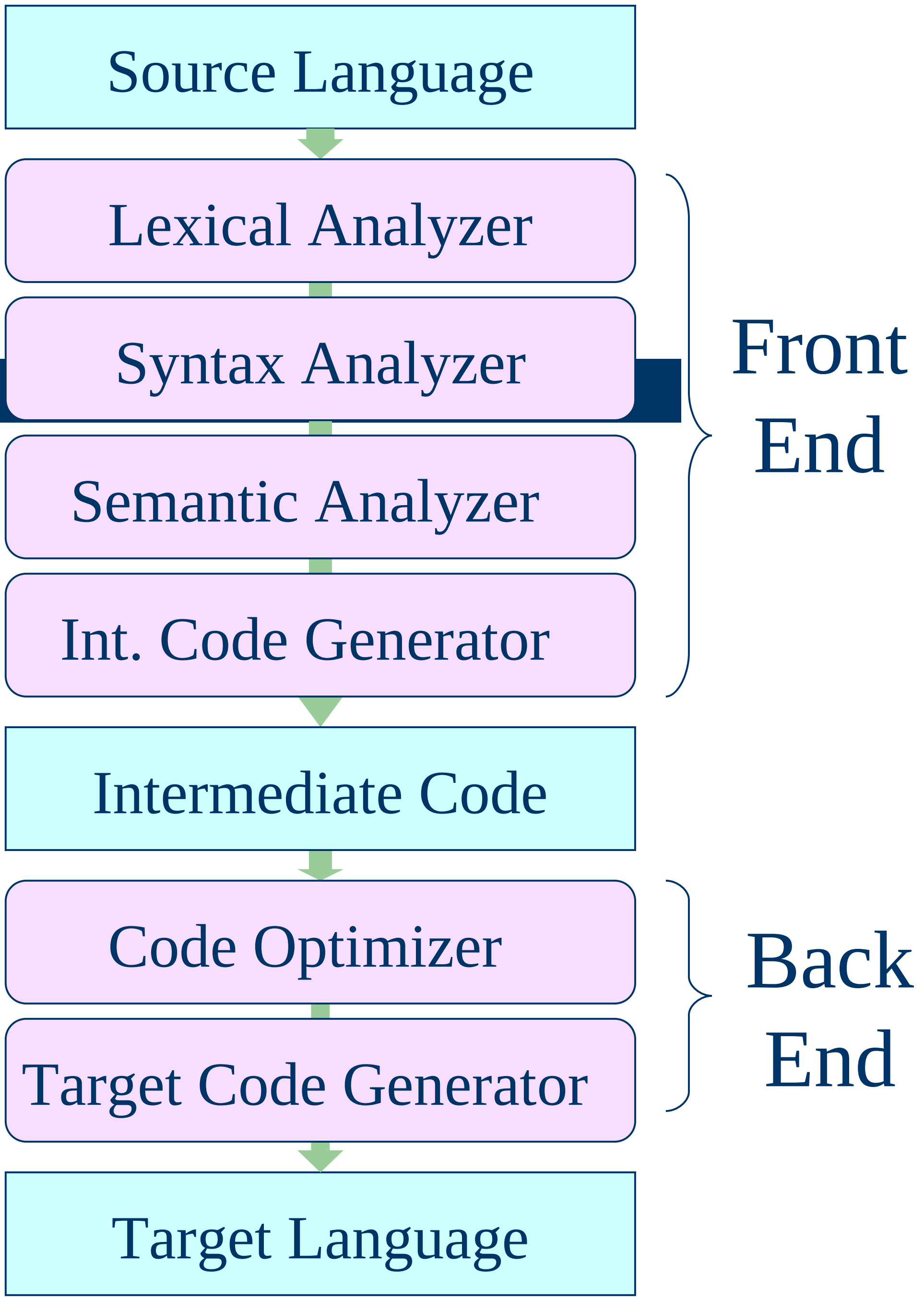
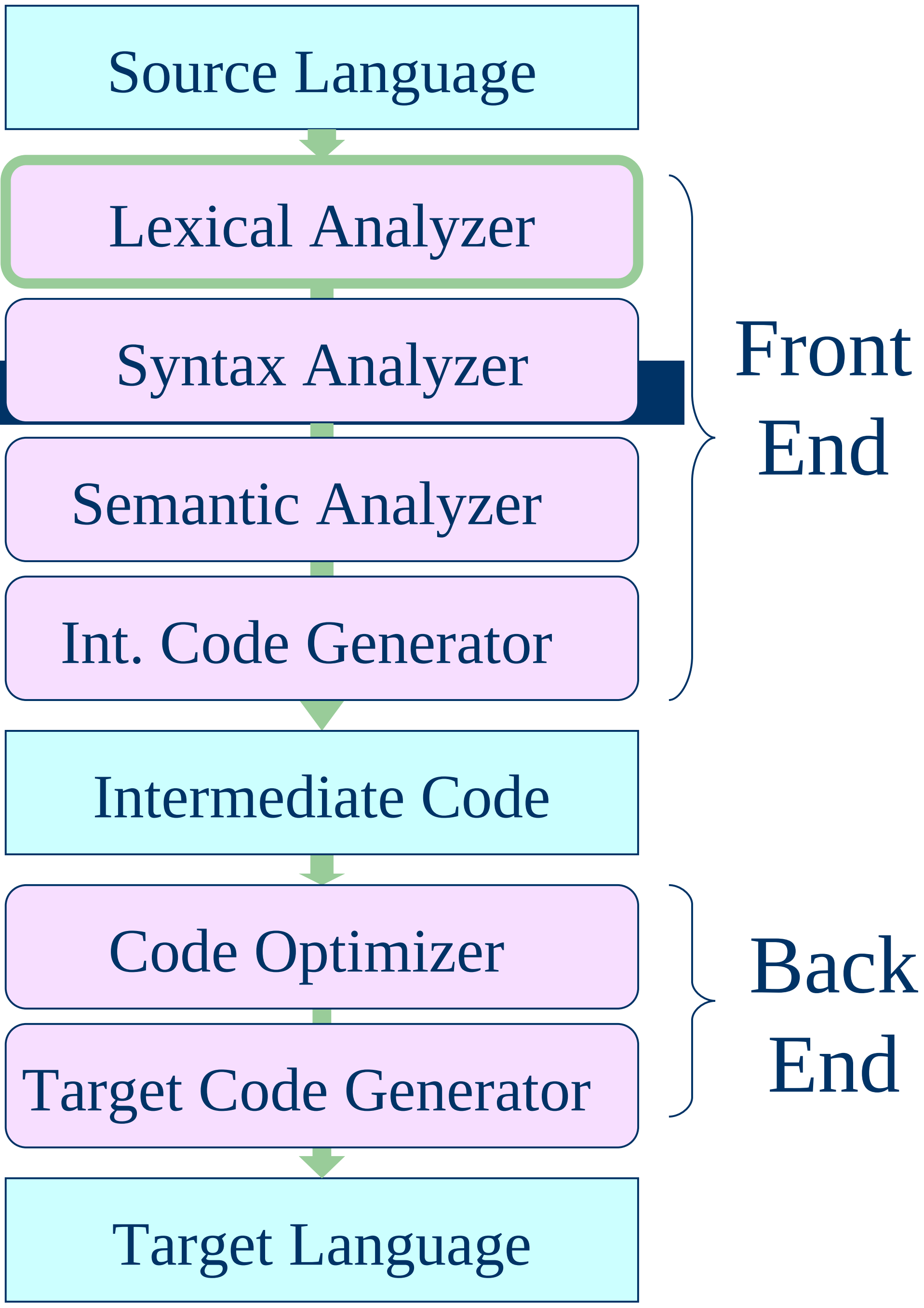
Lexical Analysis

Structure of a Compiler



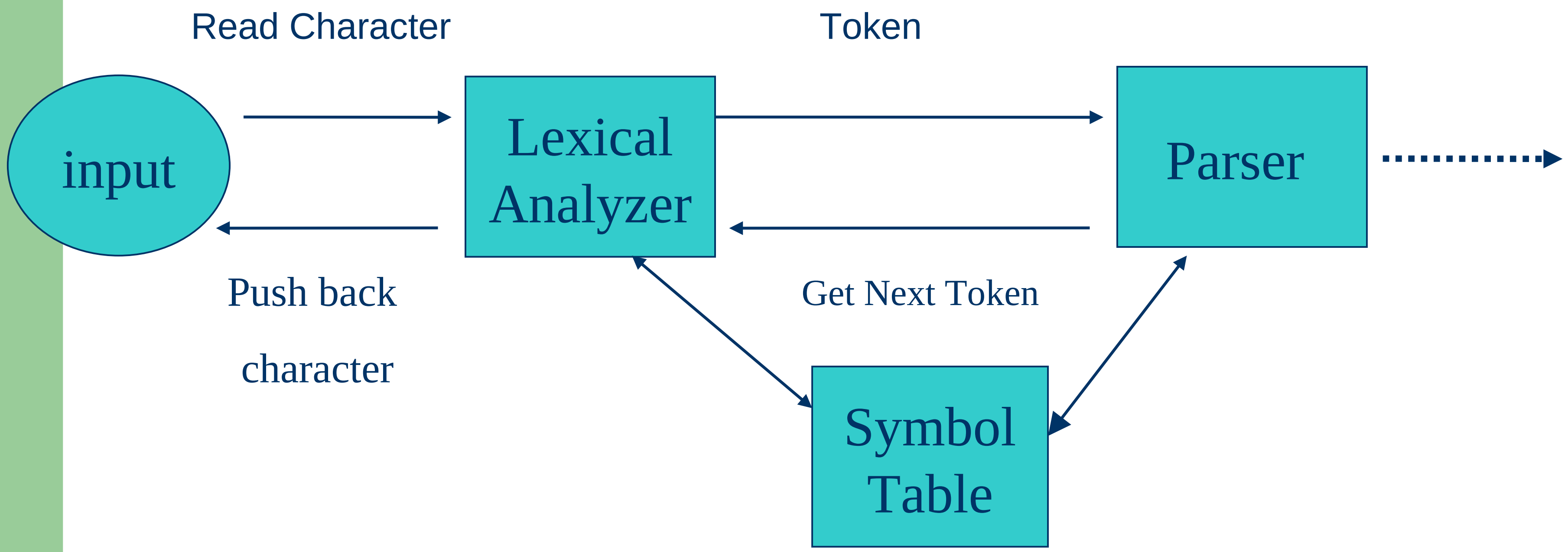
Today!


Source Language
Lexical Analyzer
Syntax Analyzer
Semantic Analyzer
Int. Code Generator
Front End
Intermediate Code
Code Optimizer
Target Code Generator
Back End
Target Language

The Role of Lexical Analyzer

The lexical analyzer is the first phase of a compiler.The main task of lexical Analyzer(Scanner) is to read a stream of characters as an input and produce a sequence of tokens that the parser(Syntax Analyzer) uses for syntax analysis.

The Role of Lexical Analyzer (cont'd)



The Role of Lexical Analyzer (cont’d)

When a lexical analyzer is inserted b/w the parser and input stream ,it interacts with two in the manner shown in the above fig.It reads characters from the input ,groups them into lexeme and passes the tokens formed by the lexemes,together with their attribute value,to the parser.In some situations ,the lexical analyzer has to read some characters ahead before it can decide on the token to be returned to the parser.

The Role of Lexical Analyzer (cont’d)

For example,a lexical analyzer for Pascal <u>must read ahead after it sees the character >.</u>If the next character is = ,then the character sequence >= is the lexeme forming the token for the “Greater than or equal to ” operator.Other wise > is the lexeme forming “Greater than ” operator ,and the lexical analyzer has read one character too many.

The Role of Lexical Analyzer (cont'd)

The extra character has to be pushed back on to the input,because it can be the beginning of the next lexeme in the input.The lexical analyzer and parser form a **producer-Consumer pair.** The lexical analyzer produces tokens and the parser consumes them.Produced tokens can be held in a token buffer until they are consumed.

The Role of Lexical Analyzer (cont'd)

The interaction b/w the two is constrained only by the size of the buffer,because the lexical analyzer can not proceed when the buffer is full and the parser can not proceed when the buffer is empty.Commonly ,the buffer holds just one token.In this case,the interaction can be implemented simply by making the lexical analyzer be a procedure called by the parser,returning tokens on demand.

The Role of Lexical Analyzer (cont'd)

The implementation of reading and pushing back character is usually done by setting up an input buffer.A block of character is read into the buffer at a time; a pointer keeps track of the portion of the input that has been analyzed.Pushing back a character is implemented by moving the pointer.

The Role of Lexical Analyzer (cont'd)

Some times lexical analyzer are divided into two phases,the first is called **Scanning** and the second is called **Lexical Analysis**.The scanning is responsible for doing simple tasks ,while the lexical analyzer does the more complex operations.For example, a Fortran might use a scanner to eliminate blanks from the input.

Issues in Lexical Analysis

There are several reasons for separating the analysis phase of compiling into lexical analysis and parsing.

- **Simpler design** is perhaps the most important consideration. The separation of lexical analysis from syntax analysis often allows us to simplify one or other of these phases. For example, a parser representing the conventions for comments and white space is significantly more complex---

Issues in Lexical Analysis (cont'd)

--then one that can assume comments and white space have already been removed by a lexical analyzer.If we are designing a new language,separating the lexical and syntactic conventions can lead to a cleaner over all language design.

Issues in Lexical Analysis (cont'd)

2) **Compiler efficiency is improved**.

A separate lexical analyzer allows us to construct a specialized and potentially more efficient processor for the task.A large amount of time is spent reading the source program and partitioning it into tokens.Specialized buffering techniques for reading in put characters and processing tokens can significantly speed up the performance of a compiler.

Issues in Lexical Analysis (cont'd)

3) **Compiler portability is enhanced**.

Input alphabet peculiarities and other device-specific anomalies can be restricted to the lexical analyzer.The representation of special or non- standard symbols,such as ↑ in Pascal ,can be isolated in the lexical analyzer.

Issues in Lexical Analysis (cont'd)

Specialized tools have been designed to help automate the construction of lexical analyzers and parsers when they are separated.

What exactly is lexing?

Consider the code:

```
if (i==j);
     z=1;
else;
     z=0;
endif;
```

i	f	_	(	i	=	=	j	)	;	\n	\t	z	=	1	;	\n	e	l	s	e	;	\n	\t	z	=	0	;	\n	e	n	d	i	f	;

This is really nothing more than a string of characters:
During our lexical analysis phase we must divide this string into meaningful sub-strings.

Tokens

- The output of our lexical analysis phase is a streams of tokens.
- A token is a syntactic category.
- In English this would be types of words or punctuation, such as a “noun”, “verb”, “adjective” or “end-mark”.
- In a program, this could be an “identifier”, a “floating-point number”, a “math symbol”, a “keyword”, etc…

Identifying Tokens

- A sub-string that represents an instance of a token is called a lexeme.
- The class of all possible lexemes in a token is described by the use of a pattern.
- For example, the pattern to describe an identifier (a variable) is a string of letters, numbers, or underscores, beginning with a non-number.
- Patterns are typically described using regular expressions.

Implementation

A lexical analyzer must be able to do three things:

1. Remove all whitespace and comments.

2. Identify tokens within a string.

3. Return the lexeme of a found token, as well as the line number it was found on.

Example

```
if_(i==j);\n\tz=1;\nelse;\n\tz=0;\nendif;
```

Line	Token		Lexeme
1	BLOCK_COMMAND		if
1	OPEN_PAREN	(	
1	ID		i
1	OP_RELATION	==	
1	ID		j
1	CLOSE_PAREN	)	
1	ENDLINE	;	
2	ID		z
2	ASSIGN	=	
2	NUMBER	1	
2	ENDLINE	;	
3	BLOCK_COMMAND		else

- Etc…

Lookahead

- Lookahead will typically be important to a lexical analyzer.
- Tokens are typically read in from left-to-right, recognized one at a time from the input string.
- It is not always possible to instantly decide if a token is finished without looking ahead at the next character. For example…
- Is “i” a variable, or the first character of “if”?
- Is “=” an assignment or the beginning of “==”?

TOKENS

The output of our lexical analysis phase is a streams of tokens.A token is a syntactic category.

"A name for a set of input strings with related structure"

Example: "ID," "NUM", "RELATION","IF"

In English this would be types of words or punctuation, such as a "noun", "verb", "adjective" or "end-mark".

In a program, this could be an "identifier", a "floating-point number", a "math symbol", a "keyword", etc…

Tokens (cont’d)

As an example,consider the following line of code,which could be part of a “ C ” program.

```
a [index] = 4 + 2
```

Tokens (cont'd)

a	ID
[	Left bracket
index	ID
]	Right bracket
=	Assign
4	Num
+	plus sign
2	Num

Tokens (cont'd)

Each token consists of one or more characters that are collected into a unit before further processing takes place.

A scanner may perform other operations along with the recognition of tokens.For example,it may enter identifiers into the symbol table.

Attributes For Tokens

The lexical analyzer collects information about tokens into their associated attributes.As a practical matter ,a token has usually only a single attribute, a pointer to the symbol-table entry in which the information about the token is kept;the pointer becomes the attribute for the token.

Attributes For Tokens (cont’d)

Let **num** be the token representing an integer. when a sequence of digits appears in the input stream,the lexical analyzer will pass **num** to the parser.The value of the integer will be passed along as an attribute of the token **num**.Logically, the lexical analyzer passes both the token and the attribute to the parser.

Attributes For Tokens (cont'd)

If we write a token and its attribute as a tuple enclosed b/w < >, the input

33 + 89 – 60

is transformed into the sequence of tuples

< **num**, 33 > <+, > <**num**, 89 > <-, > <**num**, 60>

Attributes For Tokens (cont’d)

The token “+” has no attribute ,the second components of the tuples ,the attribute ,play no role during parsing,but are needed during translation.

Attributes For Tokens (cont’d)

The token and associated attribute values in the “ C ” statement.

E = M + C * 2

Tell me the token and their attribute value ?

Attributes For Tokens (cont’d)

< ID , pointer to symbol-table entry for E >

< assign_op , >

< ID , pointer to symbol entry for M >

< add_op , >

< ID , pointer to symbol entry for C >

< mult_op , >

< num ,integer value 2 >

Pattern

"The set of strings described by a rule called a pattern associated with the token."

The pattern is said to match each string in the set.

For example, the pattern to describe an identifier (a variable) is a string of letters, numbers, or underscores, beginning with a non-number.

The pattern for identifier token, ID, is

ID → letter (letter | digit)*

Patterns are typically described using regular expressions.(RE)

Lexeme

"A lexeme is a sequence of characters in the source program that is matched by the pattern for a token"

For example, the pattern for the Relation_op token contains six lexemes (=, < >, <, < =, >, >=) so the lexical analyzer should return a relation_op token to parser whenever it sees any one of the six.

Example

Input: `count = 123`

Tokens:

identifier : Rule: “letter followed by …”

Lexeme: count

assg_op : Rule: =

Lexeme: =

integer_const : Rule: “digit followed by …”

Lexeme: 123

Tokens,Patterns.Lexemes

Token	Sample Lexemes	Informal Description of Pattern
const	const	const
if	if	if
relation	<, <=, =, < >, >, >=	< or <= or = or < > or >= or >
id	pi, count, D2	letter followed by letters and digits
num	3.1416, 0, 6.02E23	any numeric constant

Classifies
Pattern

Tokens,Patterns.Lexemes (cont'd)

Example:

Const pi = 3.1416;

In the example above ,when the character sequence "pi" appears in the source program,a token representing an identifier is retuned to the parser(ID).The returning of a token is often implemented by passing an integer corresponding to the token.

THANKS