



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Experiment in Compiler Construction Introduction to compilers

School of Information and Communication
Technology

Hanoi University of Science and
Technology

Introduction to compiler

- Compiler vs interpreter
- Phases of a compiler

Compiler vs interpreter

- Carry out the same purpose: translating high level language instructions into the binary form that is understandable by the computer.
- Interpreter: reads a statement (instruction), translates and then executes it, then take another
- Compiler: translates the entire program in one go and executes it (through .exe file)

Language examples

Compiled

C, C++, Objective-C

Interpreted

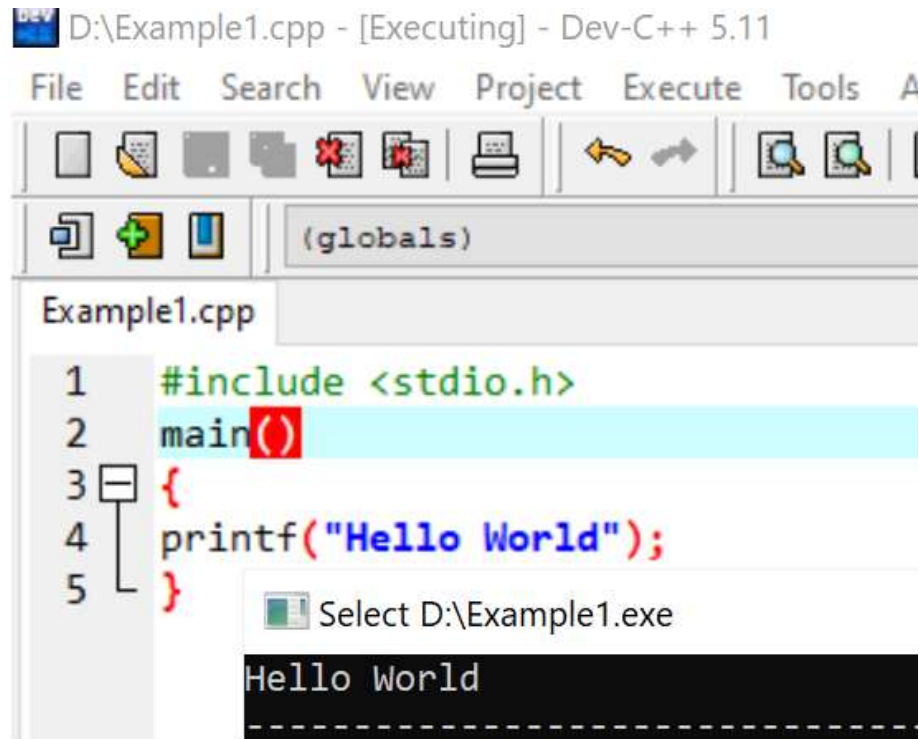
PHP, JavaScript

Hybrid

Java, C#, VB.NET, Python



Examples of Hello World program

C language

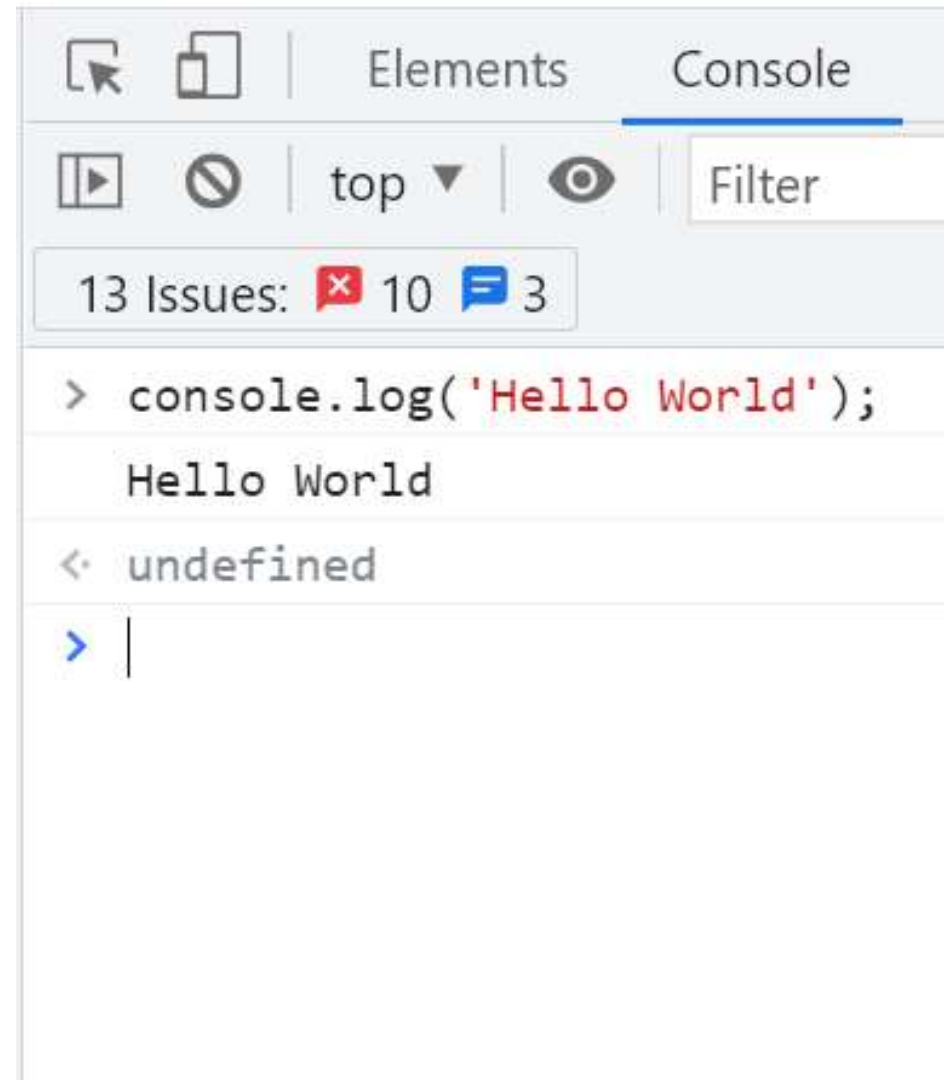


```
D:\Example1.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools A
(globals)
Example1.cpp
1 #include <stdio.h>
2 main()
3 {
4     printf("Hello World");
5 }
Select D:\Example1.exe
Hello World
```

This PC > New Volume (D:)

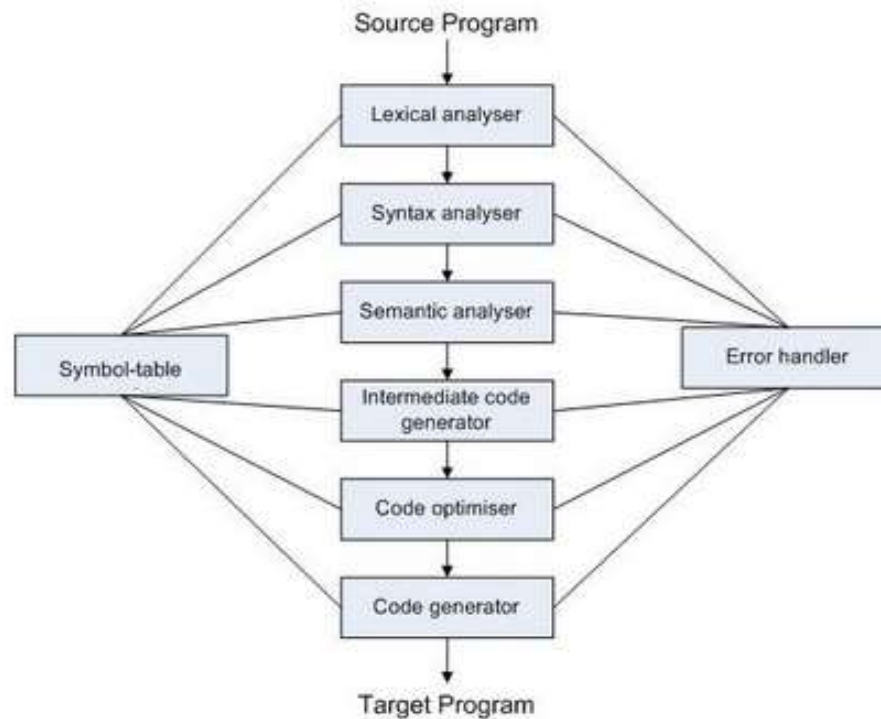
Name	Date modified	Type
 Example1	22-Sep-21 2:57 PM	C++ So
 Example1	22-Sep-21 2:57 PM	Applicat

Javascript



```
Elements Console
top Filter
13 Issues: 10 3
> console.log('Hello World');
Hello World
< undefined
> |
```

Phases of a compiler

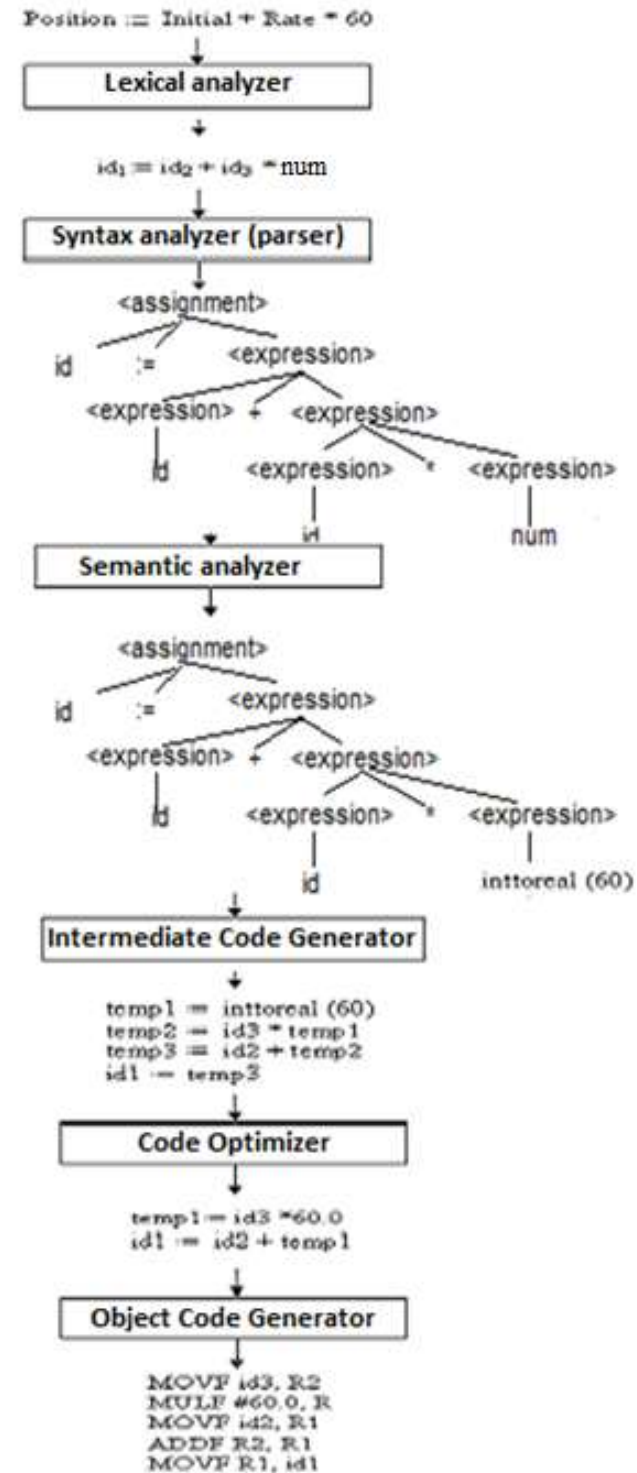


2 major phases

Analysis: Lexical analysis, syntax analysis, semantic analysis

Synthesis: Intermediate code generation, code optimization, code generation

Translation of a statement



Analysis phase

Syntax rules (grammar)

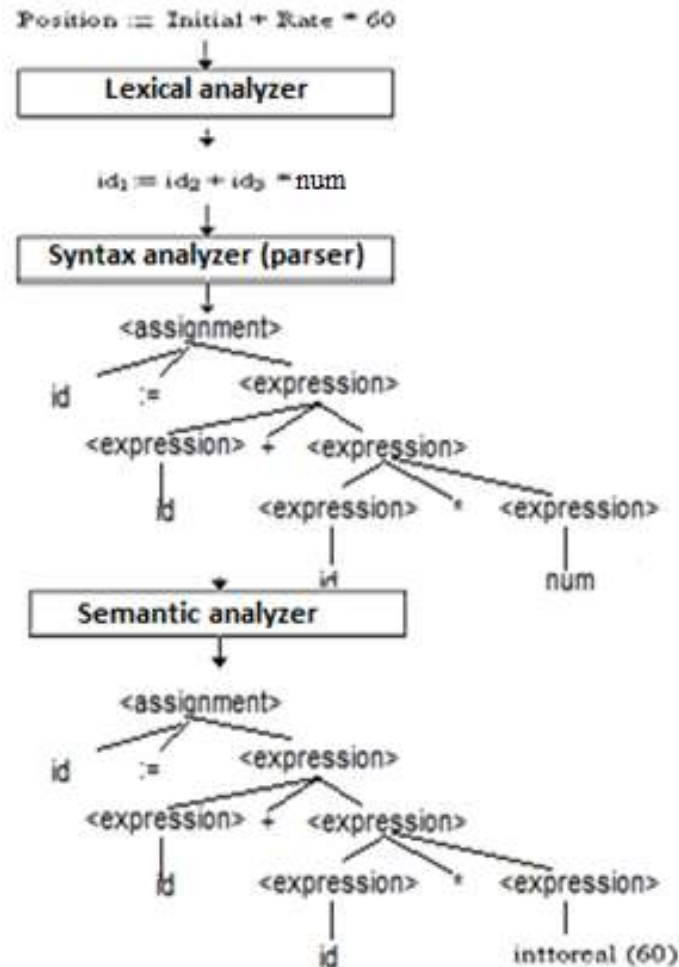
<assignment> →
id := <expression>

<expression> →
<expression> + <expression>

<expression> →
<expression> * <expression>

<expression> → id

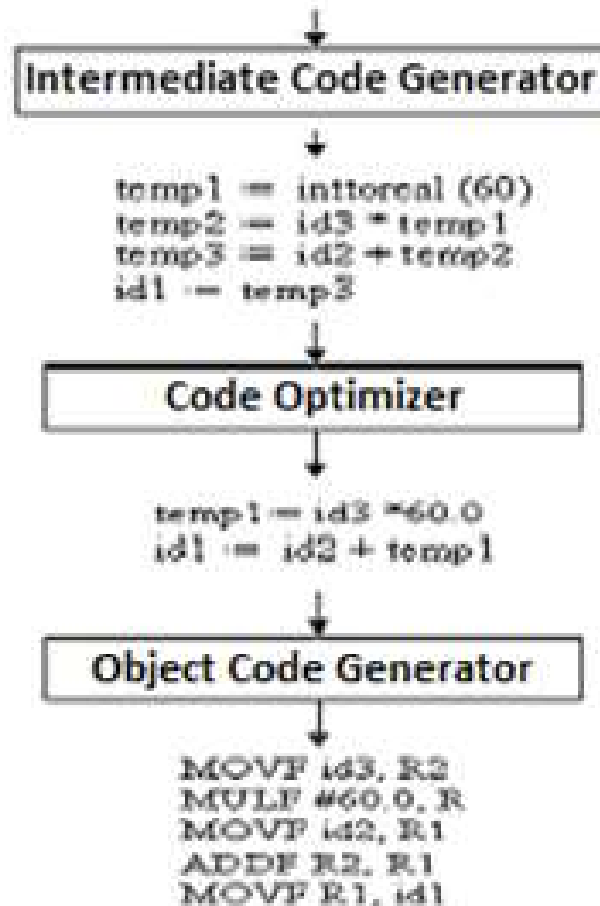
<expression> → num



Symbol table

Name	Attributes
Position	...
Initial	...
Rate	...

Synthesis phase



KPL programming language

- A tiny programming language used in writing a simple compiler
- Issued by University of Kyoto
- A subset of Pascal language

```
Program Example2; (* Factorial *)
```

```
Var n : Integer;
```

```
Function F(n : Integer) : Integer;
```

```
Begin
```

```
  If n = 0 Then F := 1 Else F := N  
* F (N - 1);
```

```
End;
```

```
Begin
```

```
  For n := 1 To 7 Do
```

```
    Begin
```

```
      Call WriteLn;
```

```
      Call WriteI(F(i));
```

```
    End;
```

```
End. (* Factorial *)
```



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

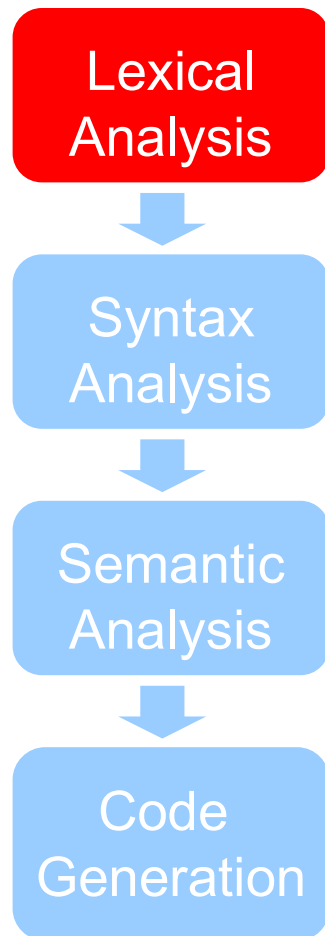
Experiment in Compiler Construction

Scanner design

School of Information and Communication
Technology

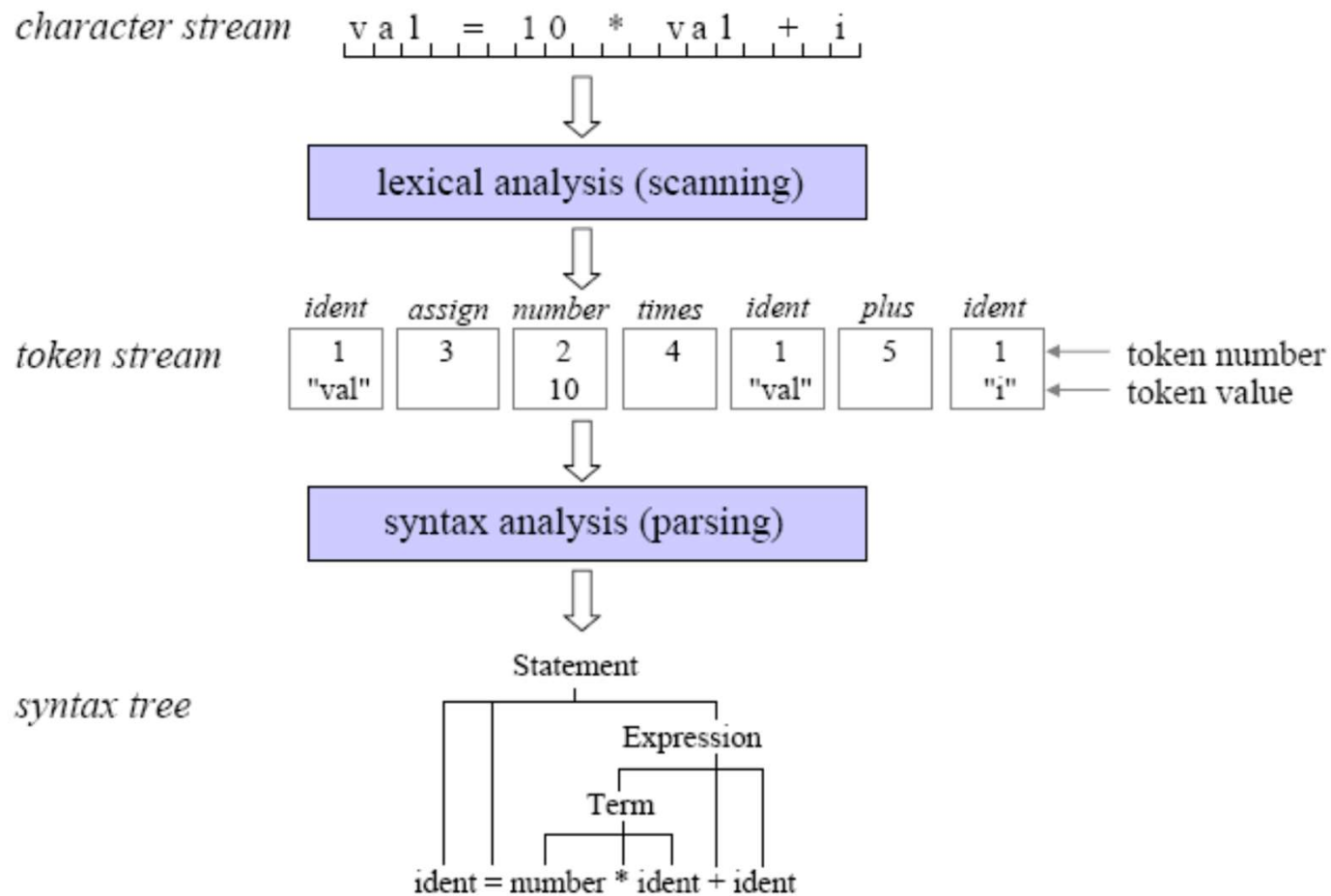
Hanoi University of Science and
Technology

What is a scanner?

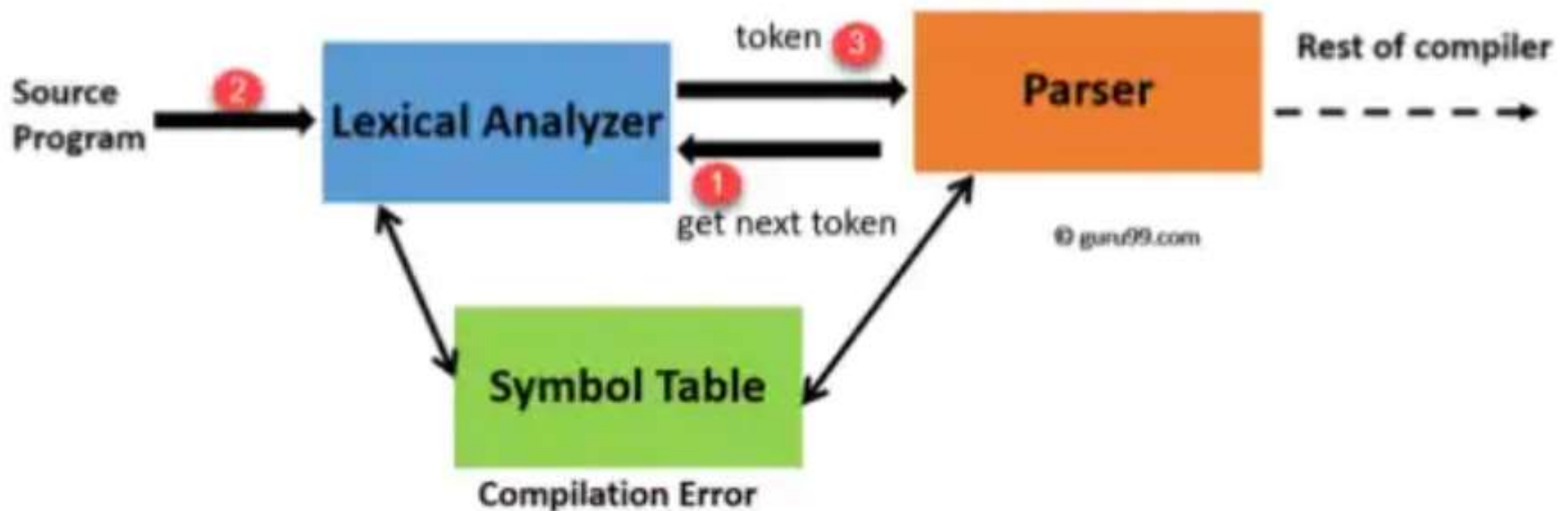


- The compiler's component/module that perform the job of lexical analysis (scanning) is called *scanner*.
- Compiler's first phase

What is a scanner?



Scanner – Parser interaction



Tasks of a scanner

- Skip meaningless characters: blank, tab, new line character, comment.
- Recognize illegal character
- Return error message
- Recognize different types of token
 - identifier
 - keyword
 - number
 - special symbols
 - ...

Tasks of a scanner

- Recognize tokens of different types
 - identifier
 - keyword
 - number
 - special character
 - ...
- Pass recognized tokens to the *parser* (the module that perform the job of syntatic analysis)

Lexical rules of KPL

- Only use unsigned integer
- The KPL identifier is made with a combination of **lowercase or uppercase letters, digits**. An identifier must **start with a letter**. The length ≤ 15 .
- Only allows **character constants**. A character constant is enclosed with a pair of single quote marks. “”
- The language do not use string constant.
- - is use for subtraction only. The language does not allow unary minus and negative numbers
- The relational operator “not equal to” is represented by !=

KPL's alphabet

- Letter: a b c ... x y z
 A B C ... X Y Z
- Digit: 0 1 2 ... 8 9
- Special character:
 - + - * /
 - > < ! =
 - [space] ,(comma) . : ; ' _
 - ()

KPL's tokens

- Keywords

PROGRAM, CONST, TYPE, VAR, PROCEDURE,
FUNCTION, BEGIN, END, ARRAY, OF, INTEGER, CHAR,
CALL, IF, THEN, ELSE, WHILE, DO, FOR, TO

- Operators

:= (assign)

+ (addition), **-** (subtraction), ***** (multiplication), **/** (division)

= (comparison of equality), **!=** (comparison of difference), **>**
(comparison of greatness), **<** (comparison of lessness), **>=**
(comparison of greatness or equality), **<=** (comparison of
lessness or equality)

KPL's tokens

- Special characters

;(semicolon), . (period), : (colon), , (comma), ((left parenthesis),) (right parenthesis), ' (singlequote)

- Also

(. and .) to mark the index of an array element

(* and *) to mark the comment

- Others

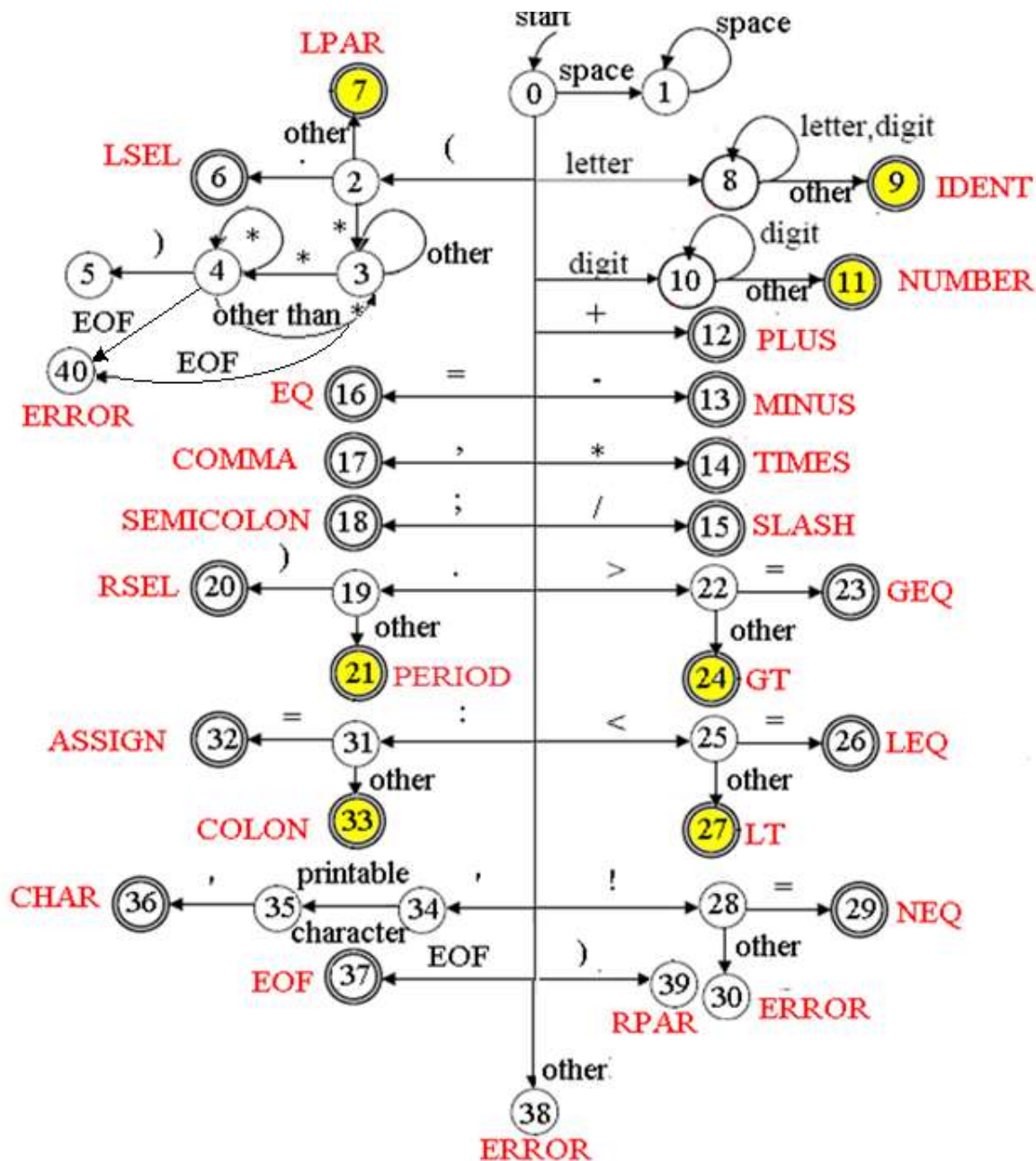
identifier, number, illegal charater

Recognizing KPL's tokens

- All KPL's tokens make up a regular language.
- They can be described with regular grammar, regular expression
- They can be recognized by a Deterministic Finite Automaton (DFA)
- The scanner is a big DFA
- The incompleted project does not use DFA. But the diagram is helpful for you to check if your scanner cover all token in KPL or not.
- Use of DFA in function skipComment also useful to process the error of unclosed comments
- I will explain how to use DFA to build a scanner in Unit 5 of the theory course. After learning that unit, you can build a new scanner project with DFA

Recognizing KPL's tokens

- After every token is recognized, the scanner starts in state 0 again
- If an illegal character is met, the scanner would change to the state -1 which tell the scanner to stop scanning and return error messages.



KPL scanner - organization

#	Filename	Task
1	Makefile	Project
2	scanner.c	Main
3	reader.h, reader.c	Read the source code
4	charcode.h, charcode.c	Classify character
5	token.h, token.c	Classify and recognize token, keywords
6	error.h, error.c	Manage error types and messages

KPL scanner – reader

```
// Read a character from input stream  
int readChar(void);
```

```
// Open input stream  
int openInputStream(char *fileName);
```

```
// Close input stream  
void closeInputStream(void);
```

```
// Current line number and column number  
int lineNo, colNo;
```

```
// Current character  
int currentChar;
```


KPL scanner – charcode

```
typedef enum {  
    CHAR_SPACE,           // space  
    CHAR_LETTER,          // character  
    CHAR_DIGIT,           // digit  
    CHAR_PLUS,            // '+'  
    CHAR_MINUS,           // '-'  
    CHAR_TIMES,           // '*'  
    CHAR_SLASH,           // '/'  
    CHAR_LT,              // '<'  
    CHAR_GT,              // '>'  
    CHAR_EXCLAMATION,     // '!'  
    CHAR_EQ,              // '='  
    CHAR_COMMA,           // ','  
    CHAR_PERIOD,          // '.'  
    CHAR_COLON,           // ':'  
    CHAR_SEMICOLON,       // ';'   
    CHAR_SINGLEQUOTE,     // '\''  
    CHAR_LPAR,            // '('  
    CHAR_RPAR,            // ')'   
    CHAR_UNKNOWN          // invalid character  
} CharCode;
```

KPL scanner – charcode

- In *charcode.c*, we define *charCodes* array that associates every ASCII character with an unique predefined *CharCode*.
- *getc()* function may return EOF (or -1) which is not an ASCII character.

KPL scanner – token

```
typedef enum {  
    TK_NONE,          // Invalid token - Error  
    TK_IDENT,         // Identifier token  
    TK_NUMBER,        // Number token  
    TK_CHAR,          // Character constant token  
    TK_EOF,           // End of program token  
    // keywords  
    KW_PROGRAM, KW_CONST, KW_TYPE, KW_VAR,  
    KW_INTEGER, KW_CHAR, KW_ARRAY, KW_OF,  
    KW_FUNCTION, KW_PROCEDURE,  
    KW_BEGIN, KW_END, KW_CALL,  
    KW_IF, KW_THEN, KW_ELSE,  
    KW_WHILE, KW_DO, KW_FOR, KW_TO,  
    // Special character  
    SB_SEMICOLON, SB_COLON, SB_PERIOD, SB_COMMA,  
    SB_ASSIGN, SB_EQ, SB_NEQ, SB_LT, SB_LE, SB_GT, SB_GE,  
    SB_PLUS, SB_MINUS, SB_TIMES, SB_SLASH,  
    SB_LPAR, SB_RPAR, SB_LSEL, SB_RSEL  
} TokenType;
```

KPL scanner – token

```
// Structure of a token
```

```
typedef struct {  
    char string[MAX_IDENT_LEN + 1];  
    int lineNo, colNo;  
    TokenType tokenType;  
    int value;  
} Token;
```

```
// Check whether a string is a keyword or not
```

```
TokenType checkKeyword(char *string);
```

```
// Create new token, provided type of token and location
```

```
Token* makeToken(TokenType tokenType, int lineNo, int  
colNo);
```

KPL scanner – error management

```
// List of error may occur in lexical analysis
typedef enum {
    ERR_ENDOFCOMMENT,
    ERR_IDENTTOOLONG,
    ERR_INVALIDCHARCONSTANT,
    ERR_INVALIDSYMBOL
} ErrorCode;

// Error message
#define ERM_ENDOFCOMMENT "End of comment expected!"
#define ERM_IDENTTOOLONG "Identification too long!"
#define ERM_INVALIDCHARCONSTANT "Invalid const char!"
#define ERM_INVALIDSYMBOL "Invalid symbol!"

// Return error message
void error(ErrorCode err, int lineNo, int colNo);
```

KPL scanner – scanner

```
// Get next token
Token* getToken(void) {
    Token *token;
    int ln, cn;

    if (currentChar == EOF)
        return makeToken(TK_EOF, lineNo, colNo);

    switch (charCodes[currentChar]) {
    case CHAR_SPACE: skipBlank(); return getToken();
    case CHAR_LETTER: return readIdentKeyword();
    case CHAR_DIGIT: return readNumber();
    case CHAR_PLUS:
        token = makeToken(SB_PLUS, lineNo, colNo);
        readChar();
        return token;
    case ... // more cases
```

Assignment

- Complete following function in `scanner.c`
 - `void skipBlank();`
 - `void skipComment();`
 - `Token* readIdentKeyword(void);`
 - `Token* readNumber(void);`
 - `Token* readConstChar(void);`
 - `Token* getToken(void);`

getToken() (1)

- **Program \Rightarrow getToken() \Rightarrow TokenType: token**

- digit	readNumber()
- letter	readIdentKeyword()
- blank	skipBlank()
	getToken();
- (- .	SB_LSEL
- *	skipComment()
	getToken();
- other	SB_LPAR
- ‘	readConstChar()
- < - =	SB_LE
- other	SB_LT

getToken()

(2)

- **Program \Rightarrow getToken() \Rightarrow TokenType: token**

- > -	=	SB_GE
-	other	SB_GT
- ! -	=	SB_NEQ
-	other	error: INVALIDSYMBOL
- . -)	SB_RPAR
-	other	SB_PERIOD
- : -	=	SB_ASSIGN
-	other	SB_SEMICOLON
- + - * / = , ;)		SB_...
-	other	error: INVALIDSYMBOL

getToken() (3) C code

```
case CHAR_COLON://:
    ln = lineNo;
    cn = colNo;
    readChar();
    if ((currentChar != EOF) &&
(charCodes[currentChar] == CHAR_EQ)) {
        readChar();
        return makeToken(SB_ASSIGN, ln,
cn) ;//:=
    } else return makeToken(SB_COLON,
ln, cn) ;//:=
```

readNumber()

- **readNumber() \Rightarrow TokenType: token**

readChar()

|- digit

readChar()

|- other

TK_NUMBER

- Use **atoi()** function to convert a string to an integer.

readIdentKeyword()

- **readIdentKeyword() \Rightarrow TokenType: token**
 - readChar()
 - digit, letter readChar()
count ++
 - other
 - count > MAX_IDENT_LENT
error: IDEN_TOO_LONG
 - count \leq MAX_IDENT_LENT
 - \equiv keywords KW_...
 - \neq keywords TK_IDENT

skipBlank()

- **skipBlank()**
 - | - blank
 - | - other
- readChar()
return

skipComment()

- **skipComment()**

<i>inside_comment</i>	- *	-)	return
		- other	<i>inside_comment</i>
		- EOF	error: END_OF_COMMENT
	- other		<i>inside_comment</i>
	- EOF		error: END_OF_COMMENT

readConstChar()

- **readConstChar() \Rightarrow TokenType: token**

- | - **character**

- | - **'**

TK_CHAR

- | - **other**

error: INVALID_CONST_CHAR

- | - **EOF**

error: INVALID_CONST_CHAR