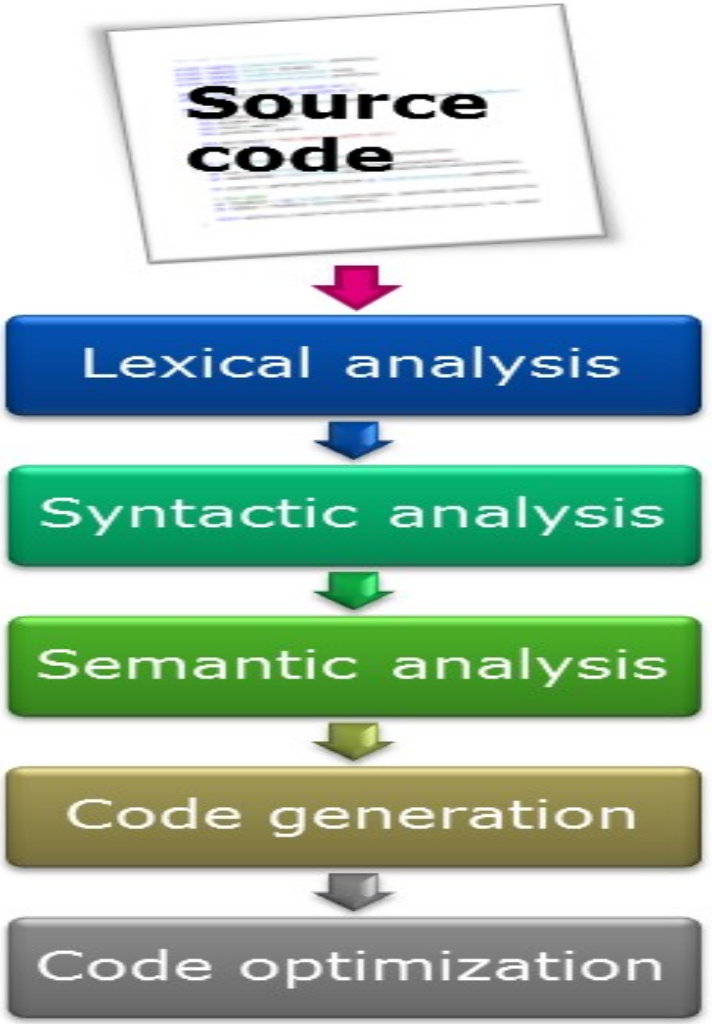
Compiler Design Laboratory (CS 653)

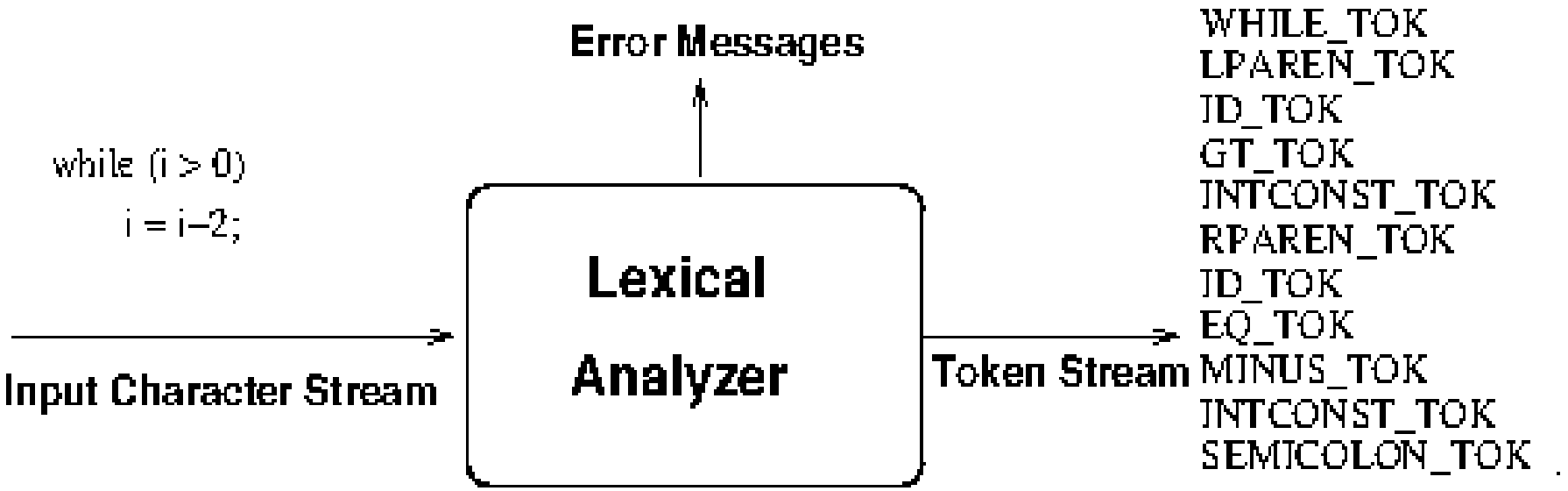
Sessional Study Materials

Manas Hira

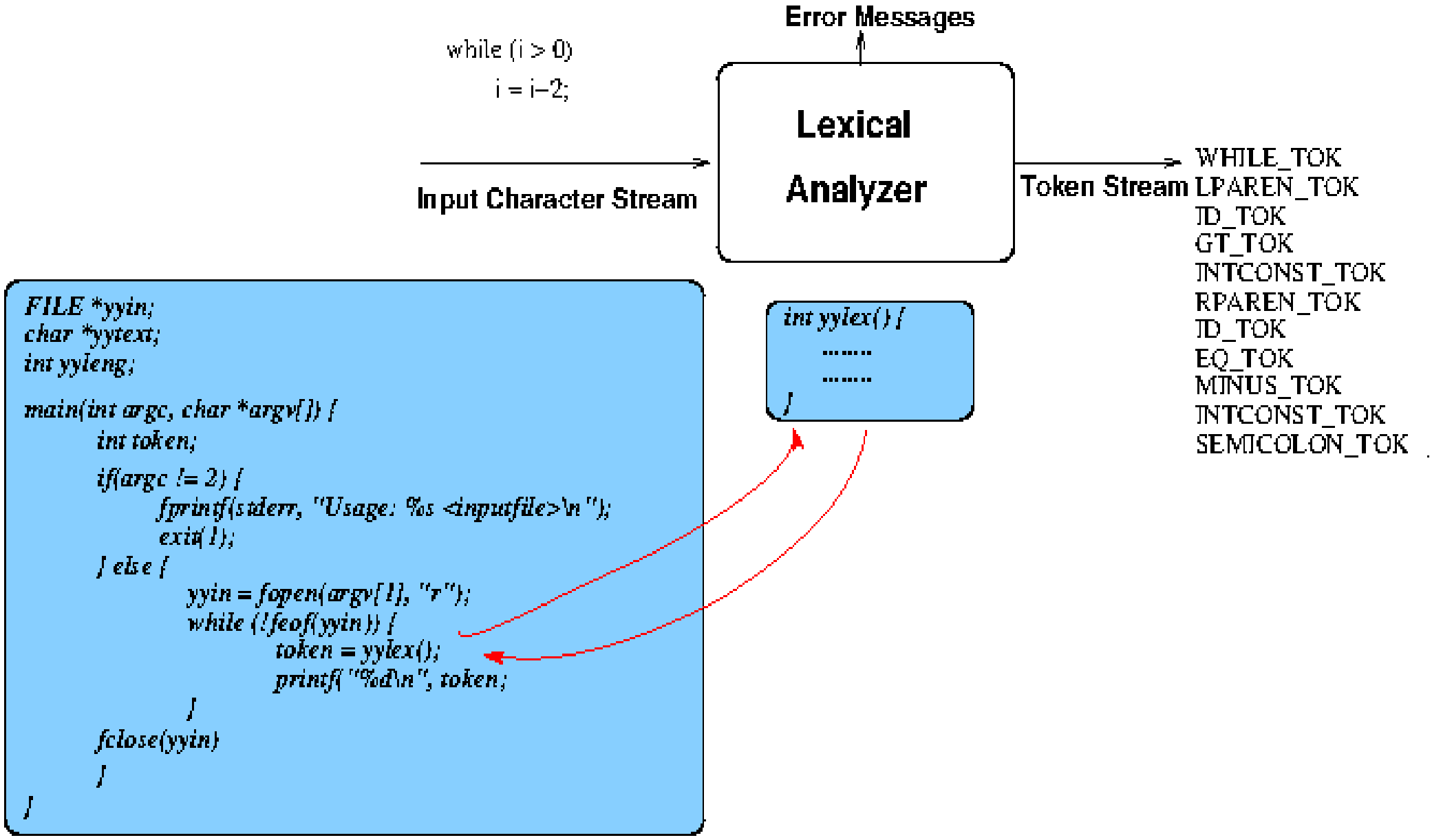
Phases of Compilation



What Lexical Analyzer does



Programmer's View



Approaches for Lexical Analysis

- Hardcoded (ad-hoc) lexical analysis – Loop and Switch approach.
- Lexical analysis based on theory of Finite Automata

Loop and switch Approach

```
/* Single caharacter lexemes */
#define LPAREN_TOK '('
#define GT_TOK '>'
#define RPAREN_TOK ')'
#define EQ_TOK '='
#define MINUS_TOK '-'
#define SEMICOLON_TOK ';'
/*
.
.
.*/
/* Reserved words */
#define WHILE_TOK 256
/*
.
.
.*/
/* Identifier, constants..*/
#define ID_TOK 350
#define INTCONST 351
/*
.
.
.*/
```

Loop and switch Approach (contd.)

```
int yylex() {
        char ch;
        If (yyin == null) {
                yyin = stdin;
        }
        ch = getc(fp); // read next char from input stream
        while (isspace(ch)) // if necessary, keep reading til non-space char
                ch = getc(fp);
                // (discard any white space)

        switch(ch) {
                case ';': case ',': case '=': // ... and other single char tokens
                yytext[0] = ch;
                yyleng = 1;
                return ch;  // ASCII value is used as token value

                case 'A': case 'B': case 'C': // ... and other upper letters
                .
                .
                case 'a': case 'b': case 'c': // ... and other lower letters
                .
                .
}
```

Assignment Statement

Implement a hardcoded lexical analyzer for **exactly** the following types of tokens

- Arithmetic, Relational, Logical, Bitwise and Assignment Operators of C
- Reserved words: for, switch-case, if-else
- Identifier
- Integer Constants
- Parentheses, Curly braces

Follow the ideas of yytext, yyleng, etc as stated in the study material.
Preferably use C++ for implementation.

Assignment Statement

Implement a hardcoded lexical analyzer for **exactly** the following types of tokens

- Arithmetic, Relational, Logical, Bitwise and Assignment Operators of C
- Reserved words: for, switch-case, if-else
- Identifier
- Integer Constants
- Parentheses, Curly braces

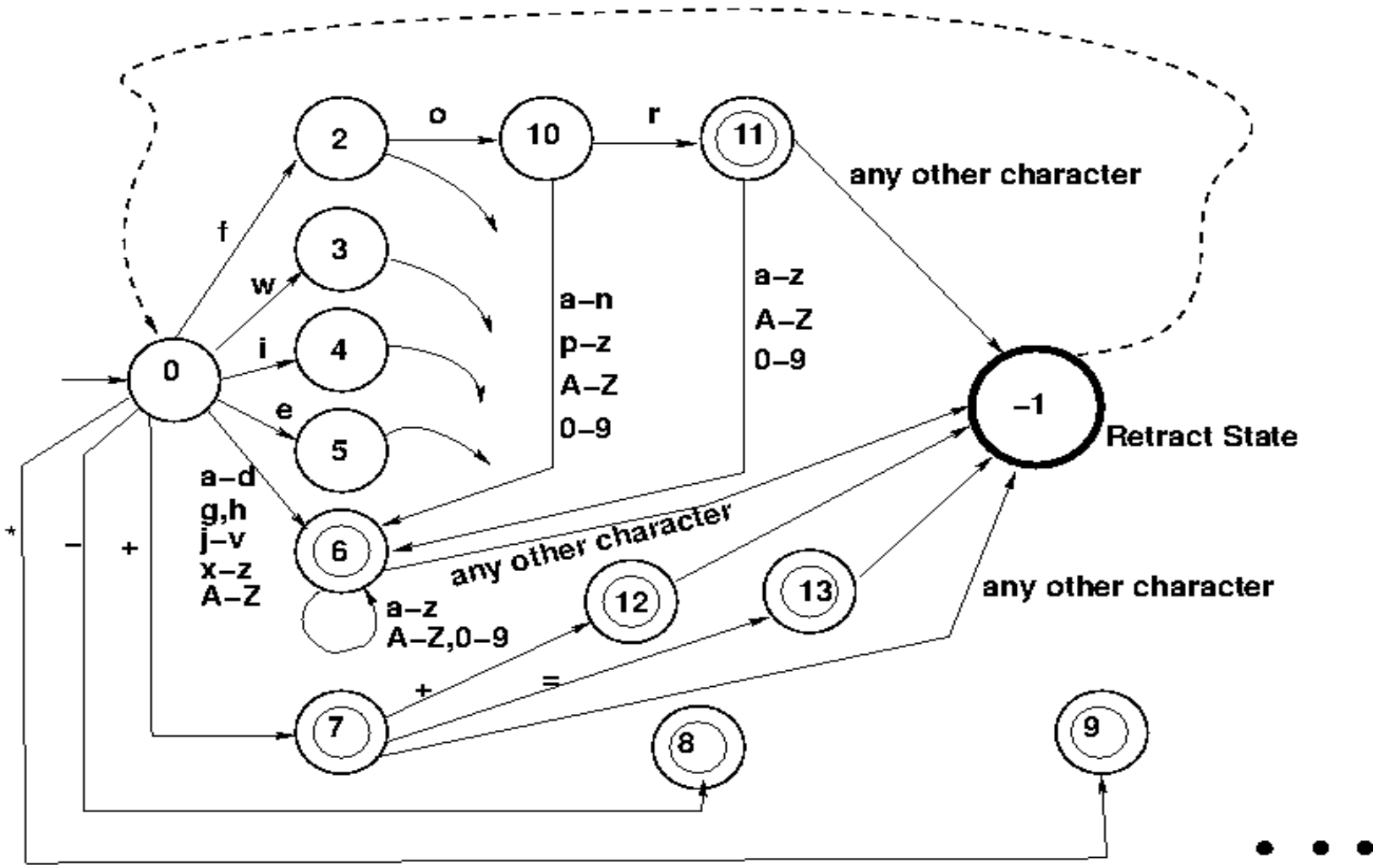
Follow the ideas of yytext, yyleng, etc as stated in the study material.
Preferably use C++ for implementation.

Lexical Analyzer based on Automata

Steps:

- Construct the Deterministic Finite Automaton (DFA) considering all the tokens of the Language (If the tokens are specified as regular expressions, then first construct the Non-Deterministic Finite Automata (NDFA) from the Regular Expressions and then convert the NDFA to an equivalent DFA.)
- Mechanically capture the DFA within the Lexical Analyzer.

Sample Finite Automata



Lexical Analyzer - Sample Code

```
int yylex() {
    int token;
    char c;
    int state;

    state = 0;
    while(1) {
        switch (state) {
            case 0:
                c = nextchar();
                if (c == 'f') state = 2;
                else if (c == 'w') state = 3;
                else if (c== 'i') state = 4;
                else if (c=='e') state = 5;
                else if ( (c >= 'a' && c <= 'd') || (c == 'g') || (c == 'h') || ...)
                    state = 6;
                else if (....
                break;
```

Lexical Analyzer - Sample Code

```
            case 2:
                c = nextchar();
                if (c == 'o') state = 10;
                else if....
                .
                .
                break;
            Case -1:
                retract();
            }
        }
}
```

Assignment Statement

Implement a lexical analyzer based on the theory of Finite Automata for **exactly** the following types of tokens

- Arithmetic, Relational, Logical, Bitwise and Assignment Operators of C
- Reserved words: for, switch-case, if-else
- Identifier
- Integer Constants
- Parentheses, Curly braces

Follow the ideas of yytext, yyleng, etc as stated in the study material.
Preferably use C++ for implementation.