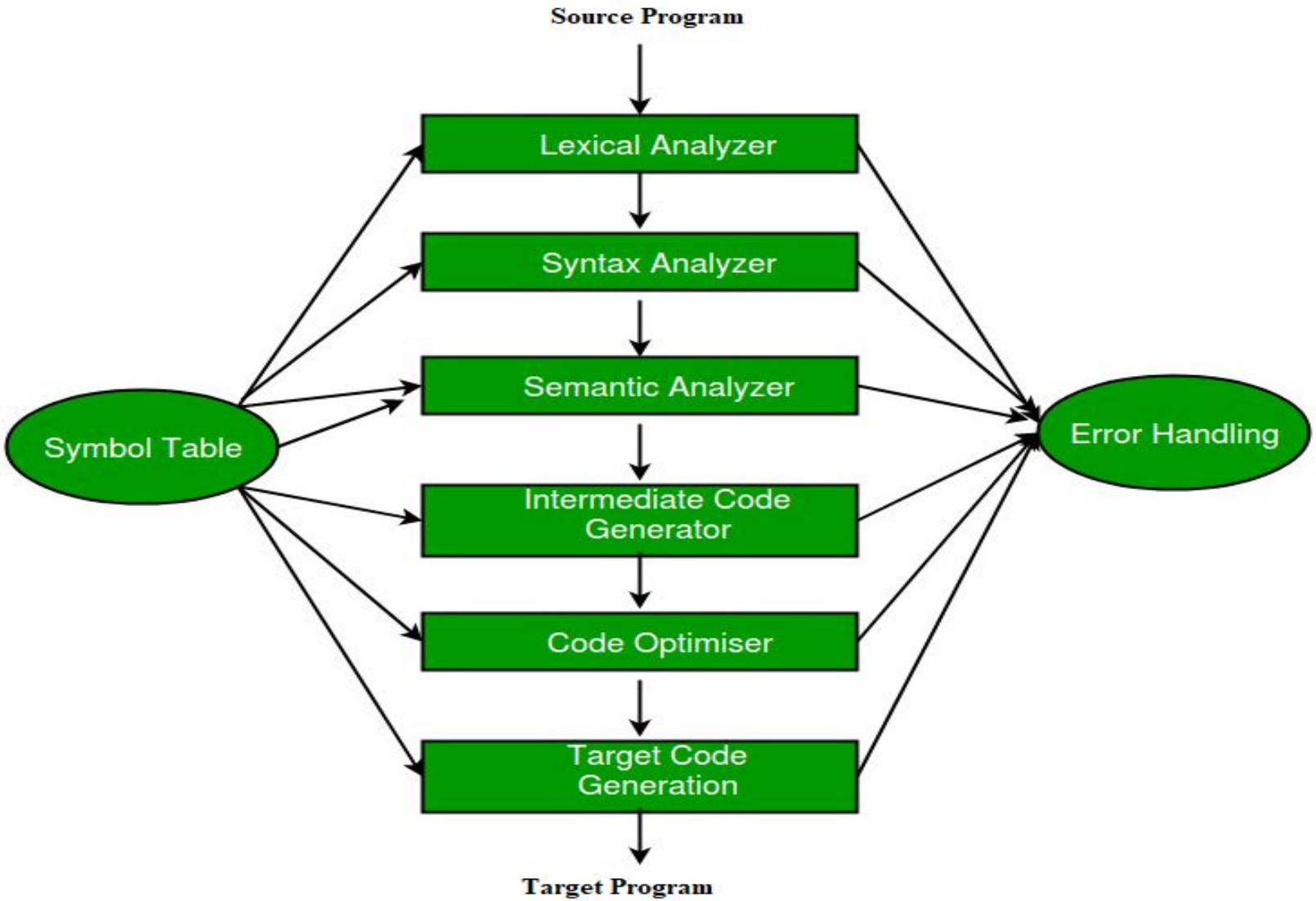
Module-2

Introduction to compilers and Lexical Analysis

What is Compiler???

Phases of Compiler



Phases of Compiler with the help of an example

position = initial + rate * 60

Phases of Compiler with the help of an example

a = b + c + d

Structure of Lex Programs

Lex program will be in following form

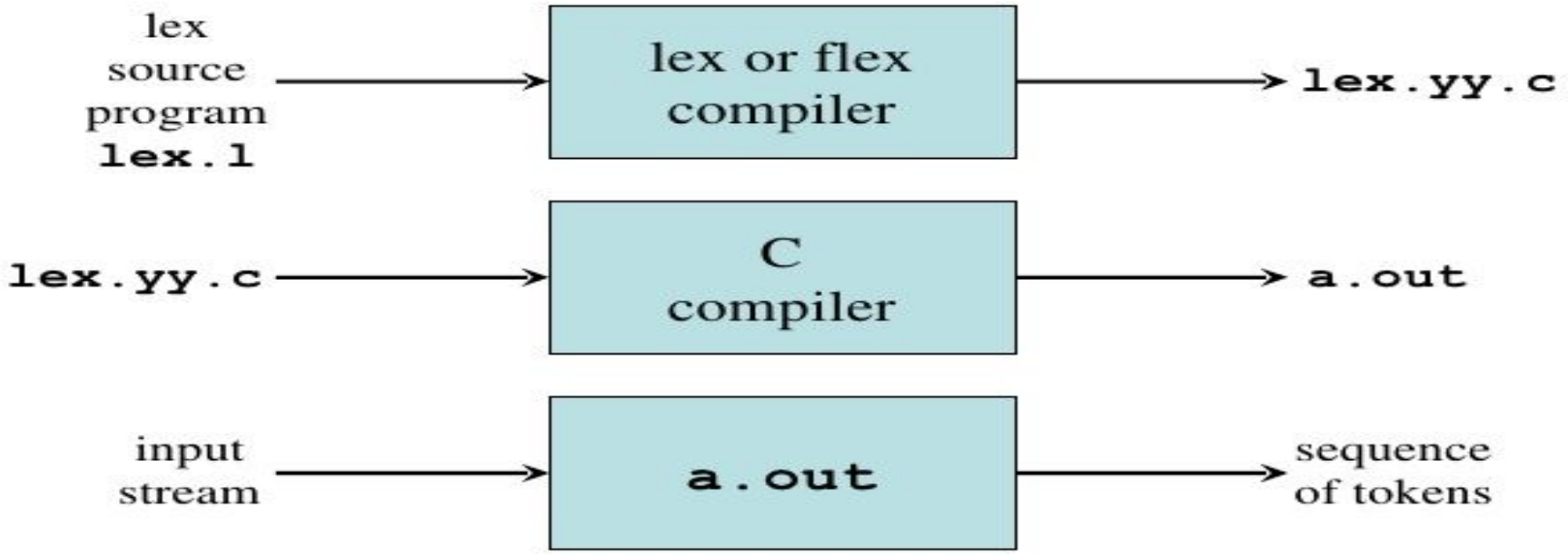
```
%{
declarations Section
%}

%%
translation rules
%%

auxiliary functions / main function
```

Structure of Lex Programs

- ***1. Declarations*** This section includes declaration of variables, constants and regular definitions.
- ***2. Translation rules*** It contains regular expressions and code segments.
- Form : Pattern {Action}
- P1 {action1}
- P2 {action2}
- ..
- …
- ..
- Pn {actionn}
- Pattern is a regular expression or regular definition.
- Action refers to segments of code.
- ***Auxiliary functions***
  - This section holds additional functions which are used in actions. These functions are compiled separately and loaded with lexical analyzer.
  - Lexical analyzer produced by lex starts its process by reading one character at a time until a valid match for a pattern is found.

Automatic Lexical Design : Lex / Flex tool: **Steps to generate LEXER**

Creating a Lexical Analyzer with Lex and Flex



Lex Program to count the number of tokens

```
%{
int n = 0 ;
%}

%%
  "while"|"if"|"else“|”int”|”float”          {n++;printf("\t keywords : %s", yytext);}
    [a-zA-Z][a-zA-Z0-9]*                     {n++;printf("\t identifier : %s", yytext);}
  "<="|"=="|"="|"++"|"-"|"*"|"+"             {n++;printf("\t operator : %s", yytext);}
  [(){}|, ;]                                 {n++;printf("\t separator : %s", yytext);}
  [0-9]+                                     {n++;printf("\t integer : %s", yytext);}
  ”* end *”                                  printf("\n total no. of token = %d\n", n);
%%

 int main()
 {
printf(“Enter an Expression \n");
yylex();
}
```

Phase 1 : lexical Analysis / Scanning