Course Info

Course Title: Compiler Construction
Course Code: CPCS302
Course Pre-requisite: CPCS301

Instructor Name: Dr. Rabie Ahmed,
Instructor Email: rabie.ahmed@nbu.edu.sa

Meeting: Sunday: 08:00 – 10:30
Monday: 08:00 – 10:30

Room: C107

Course Objective & Description

1. Course Objective

The Main Objective of this course is to familiarize the student with basic concepts of different phases of a of compiler theory, design, and construction.

2. Course Description

This course provides students with an investigation of compiler theory, design, and construction. It also introduces basic concepts of different phases of a compiler, which qualifies students to understand contents of this course. Topics include Compiler & Interpreter, Compilation process, Front-End and Back-End Phases of a compiler, The role of the lexical analyzer, Specification & Recognition of tokens, The role of the Syntax analyzer, Context-free grammars, Syntax-Directed Translation, Top-down & Bottom-up parsing Techniques, Finite Automata, DFA & NFA, converting Regular Expressions to Automata,

Course Learning Outcomes (CLOs)

CLOs		Aligned PLOs
1	**Knowledge and Understanding:**	
1.1	Describe the difference between compilers and interpreters in compilation process.	K1
1.2	List the phases of a compiler construction including front end and back end phases.	K2
2	**Skills:**	
2.1	Analyze the regular expressions of the specific tokens for specifying a regular language.	S1
2.2	Design a context free grammar as a syntax rule of a simple language.	S2
2.3	Apply algorithms related to front end phases of a compiler to develop the scanner and the parser of a simple language using current techniques and generator tools.	S3, S5
3	**Values:**	
3.1	Function effectively in teams to accomplish a common goal	V2

Course Assessment Tools

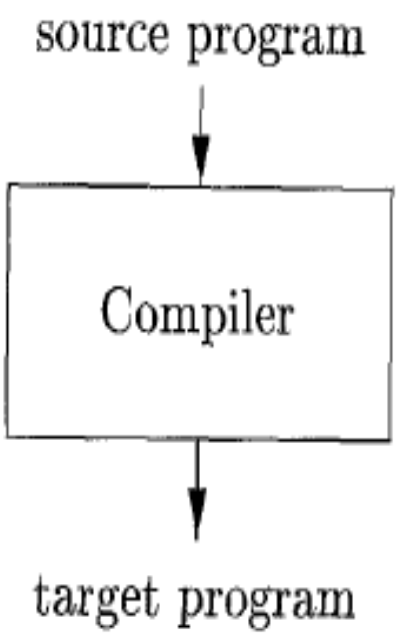
Participation:(W1-W12)	5%
Quizzes:(W2,W8)	5%
Assignments:(W4,W10)	10%
LAB Tasks:(W4-W11)	10%
LAB Exam:(W12)	10%
Mid-Term Exam:(W6)	20%
Final Exam:(W13)	40%
TOTAL:	**100%**

Course Learning Resources

Required Textbooks	1. Alfred Aho, Monica S. Lam, Ravi Sethi, and Jeffery D. Ullman **"Compilers: Principles, Techniques and Tools"** Addison Wesley, 3rd edition, 2009.
Essential References Materials	1. Cooper and Torczon. **"Engineering a Compiler",** Maurgan Kaufman, 2nd edition, 2011. 2. Dick Grune, Kees van Reeuwijk, Henri E. Bal, Ceriel J.H. Jacobs, Koen Langendoen. **"Modern Compiler Design"**, Springer, 2nd edition, 2012.
Electronic Materials	1. Blackboard System: https://lms.nbu.edu.sa/ 2. Northern Border University Electronic Library: https://www.nbu.edu.sa/AR/Deanships/Library_Issues 3. Saudi Digital Library (SDL): https://portal.sdl.edu.sa/english/

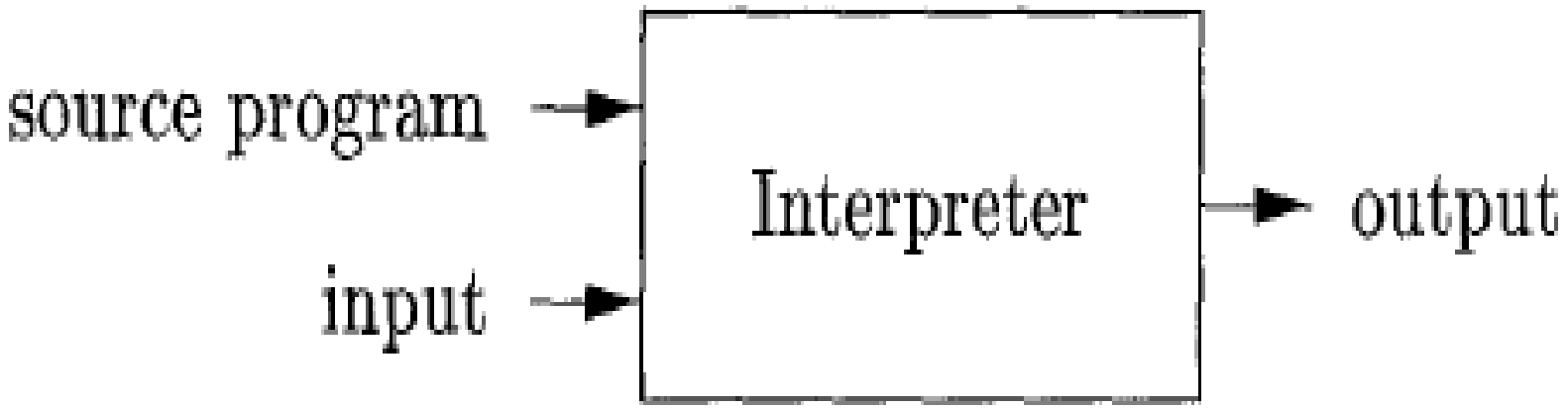
Compiler

Compiler is a program that reads a program in one language (the source language) and translates it into an equivalent program in another language (the target language)



Interpreter

Interpreter is a program that reads a program and produces the results of running that program on a given inputs



Compiler vs Interpreter

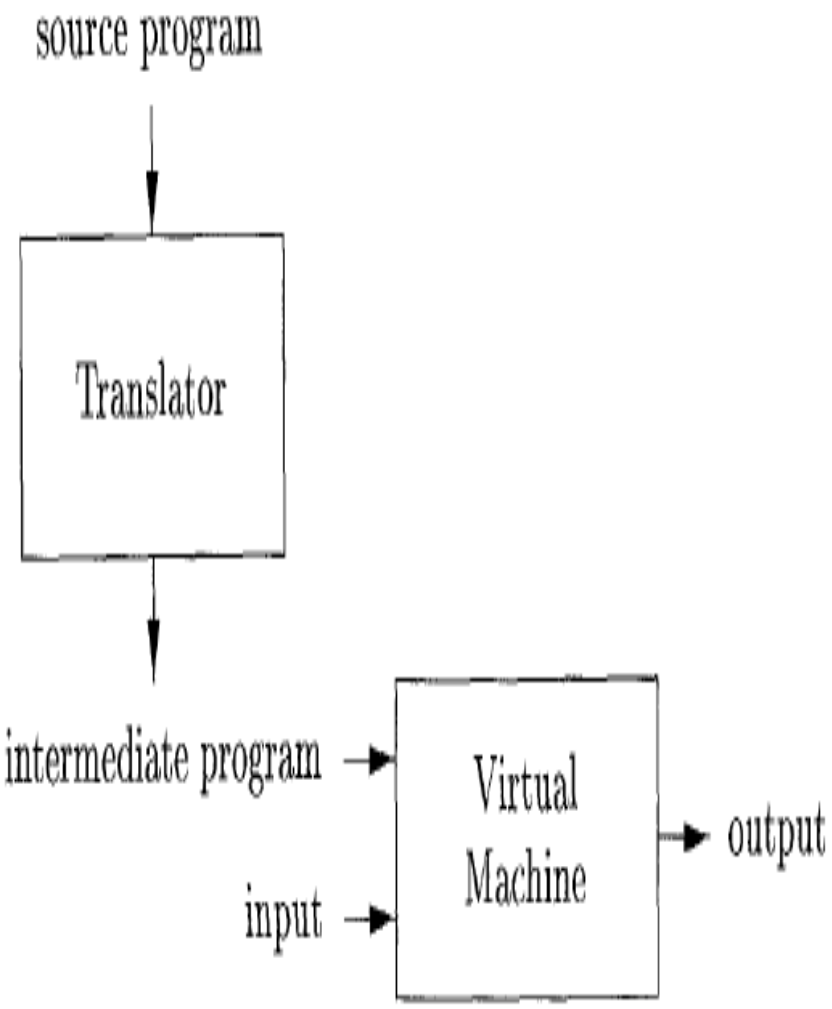
- ✓ The machine-language target program produced by a compiler is usually much faster than an interpreter at mapping inputs to outputs .
- ✓ An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement.

Compiler

Interpreter

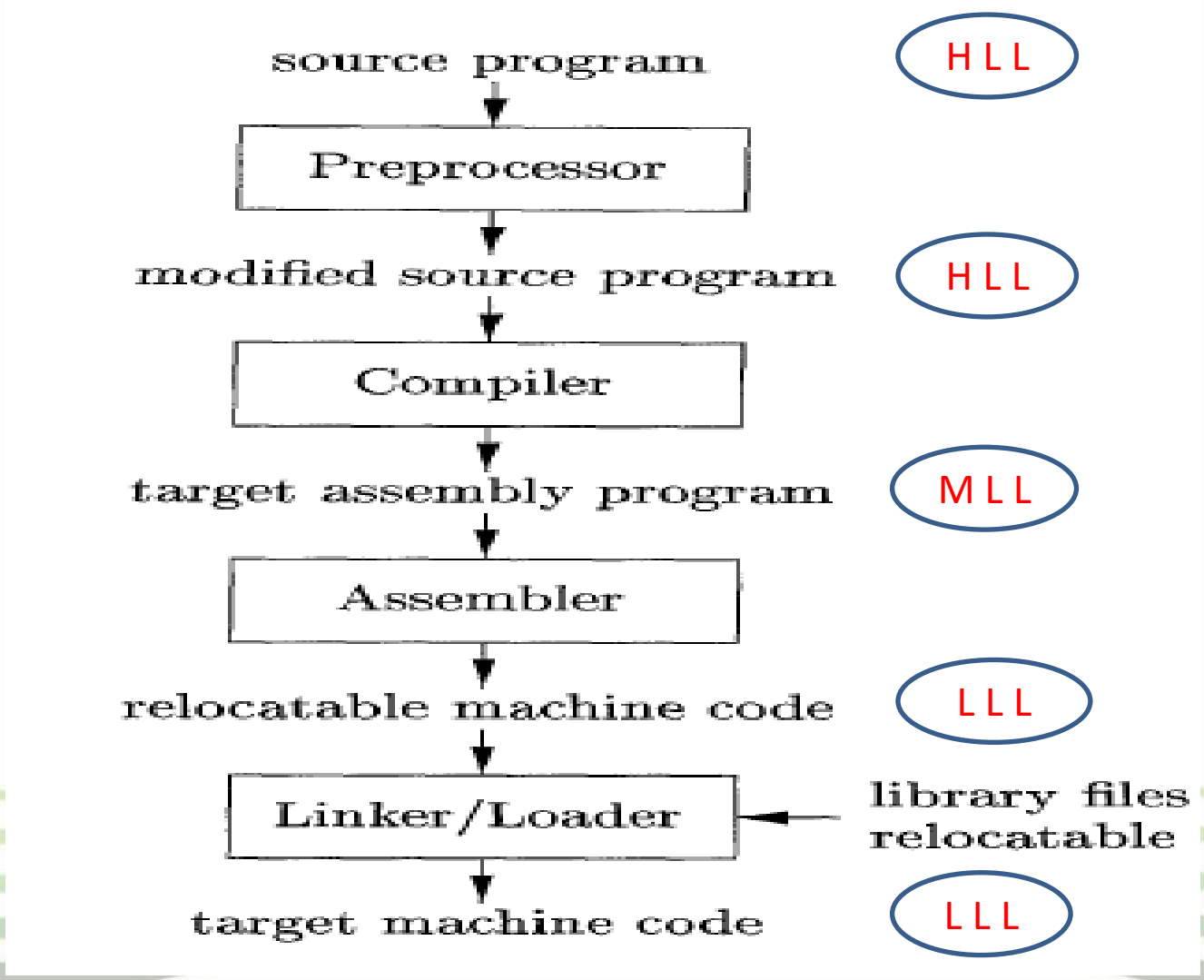
```
Int x ;
Int z ;
X := 10 ;
Y := 20 ;
Z := x + y ;
Printf << x ;
Printf << y ;
Printf << z ;
```

Hybrid Compiler

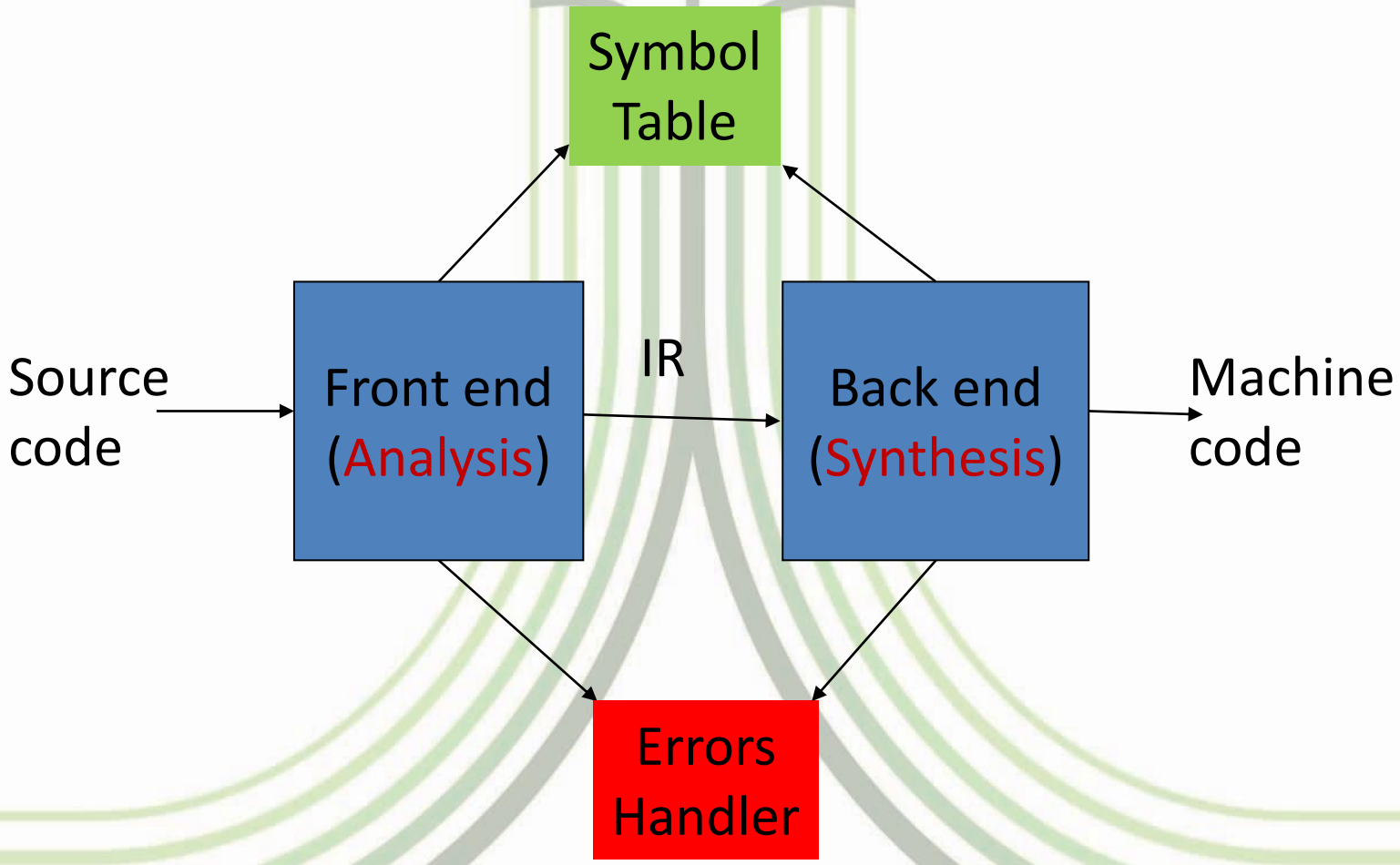


Hybrid Compiler combines compilation and interpretation, source program may first be compiled into an intermediate form called bytecodes. The bytecodes are then interpreted by a virtual machine.

Compilation Process



Front-end, Back-end division

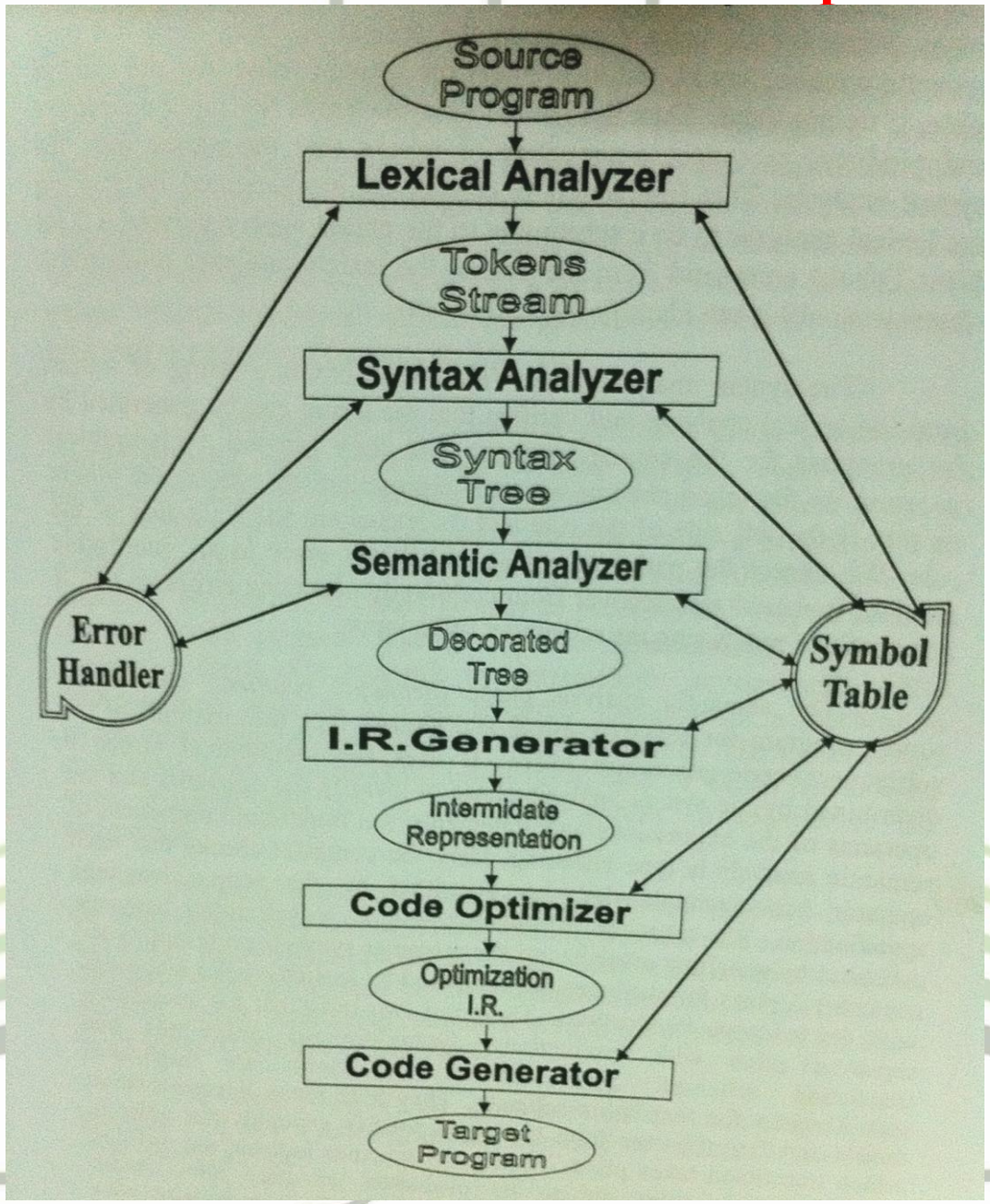


- Front end maps source code into IR
- Back end maps IR into target machine code

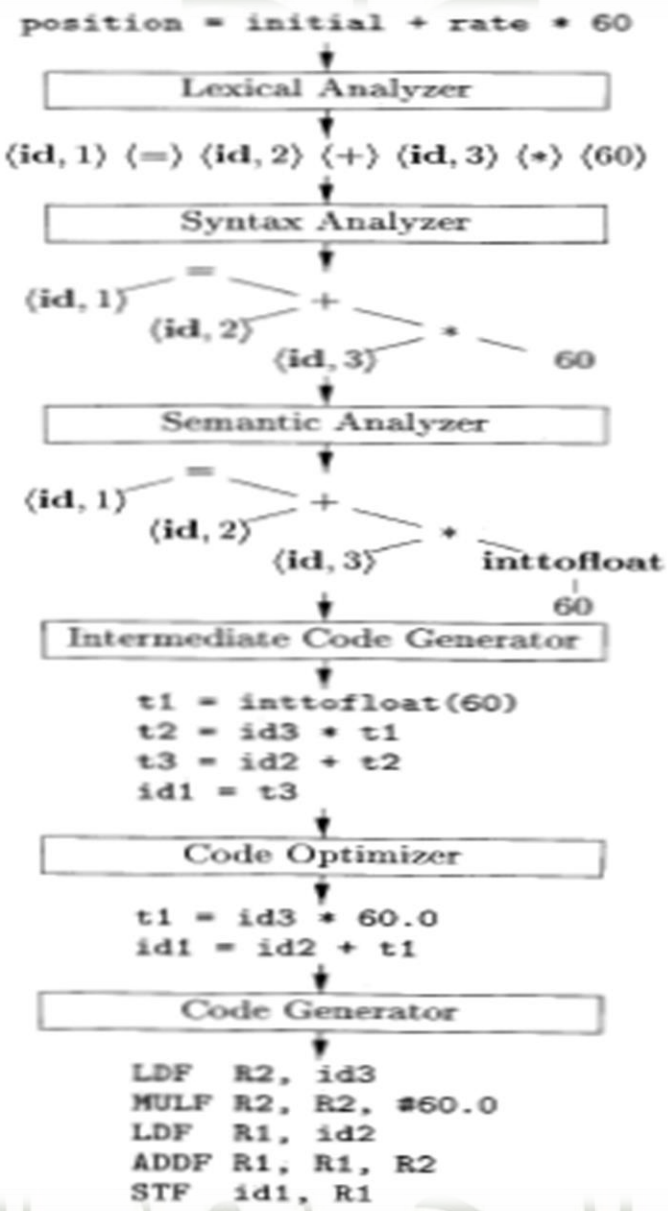
The Phases of a Compiler

- A compiler operates in phases, each of which transforms the source program from one representation to another.
- The first three phases, Lexical analysis, Syntax analysis, and Semantic analysis form analysis portion of a compiler while the last three phases Intermediate code generation, Code optimization, and Code generation form synthesis portion of a compiler.
- The analysis part creates an intermediate representation from the source program. The synthesis part constructs the target program from the intermediate representation. The analysis part is often called the front end of the compiler and the synthesis part is the back end.
- Two other activities, Symbol table management and Error handling are shown interacting with the phases.

The Phases of a Compiler



Examples on Phases of a Compiler



The Phases of a Compiler

1. Lexical Analysis

•The first phase of a compiler is called lexical analysis or scanning. The lexical analyzer reads the stream of characters making up the source program and groups the characters into meaningful sequences called lexemes.

•For each lexeme, the lexical analyzer produces as output a token of the form

<token-name, attribute-value>

The Phases of a Compiler

2. Syntax Analysis

- The second phase of the compiler is syntax analysis or parsing. The parser uses the first components of the tokens produced by the lexical analyzer to create a tree-like intermediate representation that depicts the grammatical structure of the token stream.
- A typical representation is a syntax tree in which each interior node represents an operation and the children of the node represent the arguments of the operation

The Phases of a Compiler

3. Semantic Analysis

- The semantic analyzer uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition.
- It also gathers type information and saves it in either the syntax tree or the symbol table.
- An important part of semantic analysis is type checking, where the compiler checks that each operator has matching operands.

The Phases of a Compiler

4. Intermediate Code Generation

•In the process of translating a source program into target code, a compiler may construct one or more intermediate representations, which can have a variety of forms, like, Syntax Tree or Decorated Tree or machine-like intermediate representation.

•This intermediate representation should have two important properties:

1. it should be easy to produce from semantic phase
2. it should be easy to translate into target machine.

The Phases of a Compiler

5. Code Optimization

- The machine-independent code-optimization phase attempts to improve the intermediate code so that better target code will result.
- Usually better means faster, but other objectives may be desired, such as shorter code, or target code that consumes less power.

The Phases of a Compiler

6. Code Generation

•The code generator takes as input an intermediate representation of the source program and maps it into the target language. If the target language is machine code, registers or memory locations are selected for each of the variables used by the program.

•Then, the intermediate instructions are translated into sequences of machine instructions that perform the same task.

The Phases of a Compiler

7. Symbol-Table Management

•An essential function of a compiler is to record the variable names used in the source program and collect information about various attributes of each name. These attributes may provide information about the storage allocated for a name, its type, its scope

•The symbol table is a data structure containing a record for each variable name, with fields for the attributes of the name. The data structure should be designed to allow the compiler to find the record for each name quickly and to store or retrieve data from that record quickly.

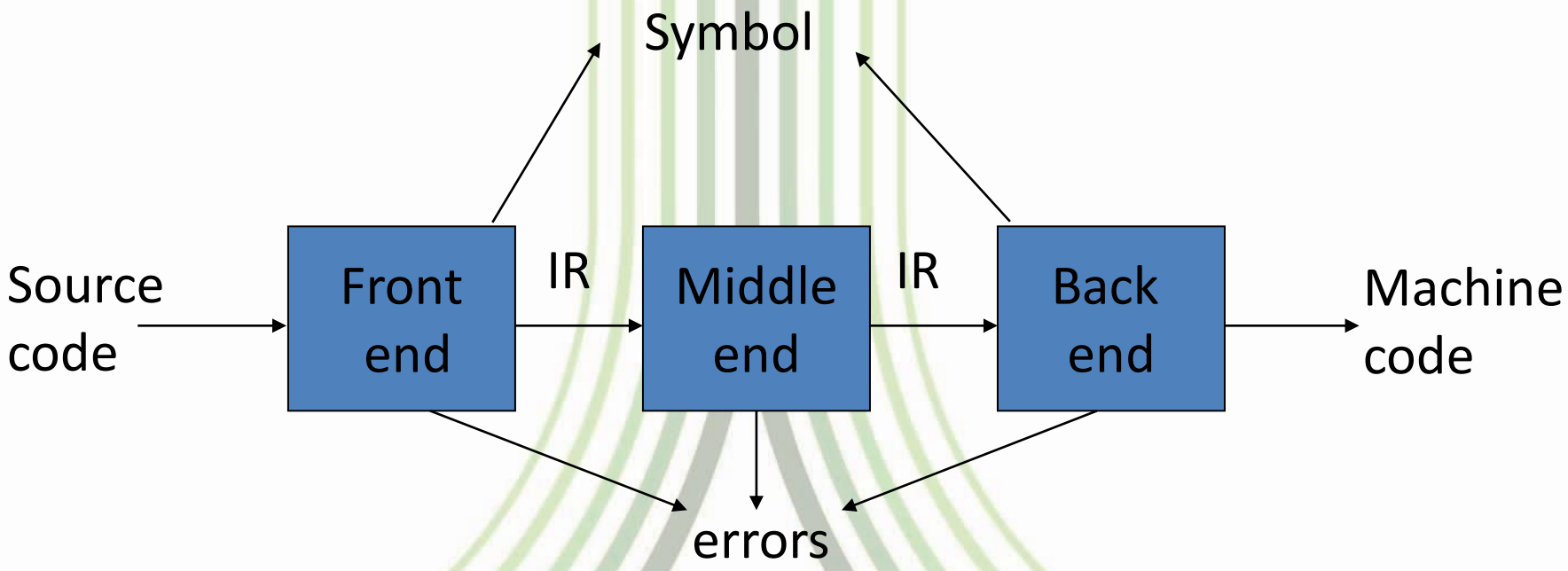
The Phases of a Compiler

8. Error Handling

•Each phase can encounter errors. However after detecting an error, a phase must somehow deal with that error, so that compilation can proceed, allowing further errors in the source program to be detected. The syntax and semantic analysis phases usually handle a large fraction of the errors detectable by the compiler.

•The lexical phase can detect errors where the characters remaining in the input do not form any token of the language. Errors where the token stream violates the structure rules of the language are determined by the syntax analysis phase. During semantic analysis the compiler detects constructs that have a right syntactic structure but no meaning to the operation involved.

Traditional three pass compiler



- Code improvement analyzes and change IR
- Goal is to reduce runtime

The Grouping of Phases into Passes

The discussion of phases deals with the logical organization of a compiler.

In an implementation, activities from several phases may be grouped together into a pass that reads an input file and writes an output file.

For example, the front-end phases of lexical analysis, syntax analysis, semantic analysis, and intermediate code generation might be grouped together into one pass.

Compiler-Construction Tools

Some commonly used compiler-construction tools include

1. *Scanner generators that produce lexical analyzers*
2. *Parser generators that automatically produce syntax analyzers*

3. Syntax-directed translation engines that produce collections of routines for walking a parse tree and generating intermediate code.

4. Data-flow analysis is a key part of code optimization.

5. *Code-generator generators that produce a code*

6. Compiler-construction toolk2ts that provide an integrated set of routines for constructing various phases of a compiler.