Course: Compiler Construction

Week 1: Introduction to Compiler Construction

Lecturer: Martha Gichuki

Lecture learning outcomes

At the end of this lecture the learner will be able to:

i) Define a compiler, interpreter and translator

ii) Describe the phases of a compiler

iii) Explain the qualities of a good compiler.

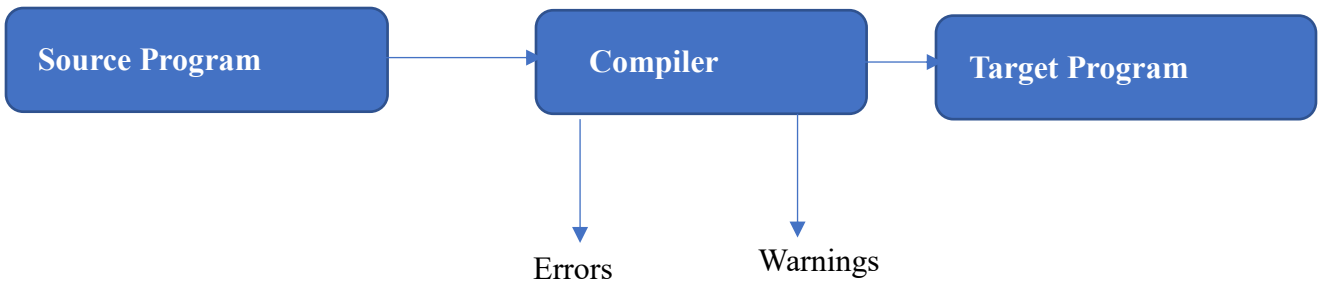
1.0 Introduction:

What is a Language Processor?

- ✓ This is a computer program that translates source code from one programming language to another[1]
- ✓ An integrated software development environment has many language processors such as compilers, interpreters, assemblers, linkers, loaders, debuggers, profilers.

What is a compiler?

- ✓ A compiler is a system software that converts a program written in high-level language that is suitable for programmers into a low-level language suitable for computers[2].
- ✓ During the process, the compiler identifies and reports obvious programmer mistakes.



[1] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27, 28

[2] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

What is an Interpreter?

This is a language processor that directly executes the operations specified in the source program on inputs supplied by the user to generate an output[3]

How does a Compiler compare to a Translator?

- ✓ Compilers that target programming languages rather than the instruction set of a computer are often called **source-to-source translators**
- ✓ Unlike Interpreters, Translators produce a target program
- ✓ Programs produced by compilers require further translation before they can execute directly on a computer
- ✓ Some languages adopt translation schemes that include both compilation and interpretation[4]

Two parts of compilation

i. ***The analysis part*** **–** breaks up the source program into constant pieces and creates *an* ***Intermediate Representation (IR)*** of the source program. *(Front-End)*

ii. ***Synthesis part*** – constructs desired target program from the Intermediate Representation. *(Back-end)*[5]

[3] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

[4] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27, 28

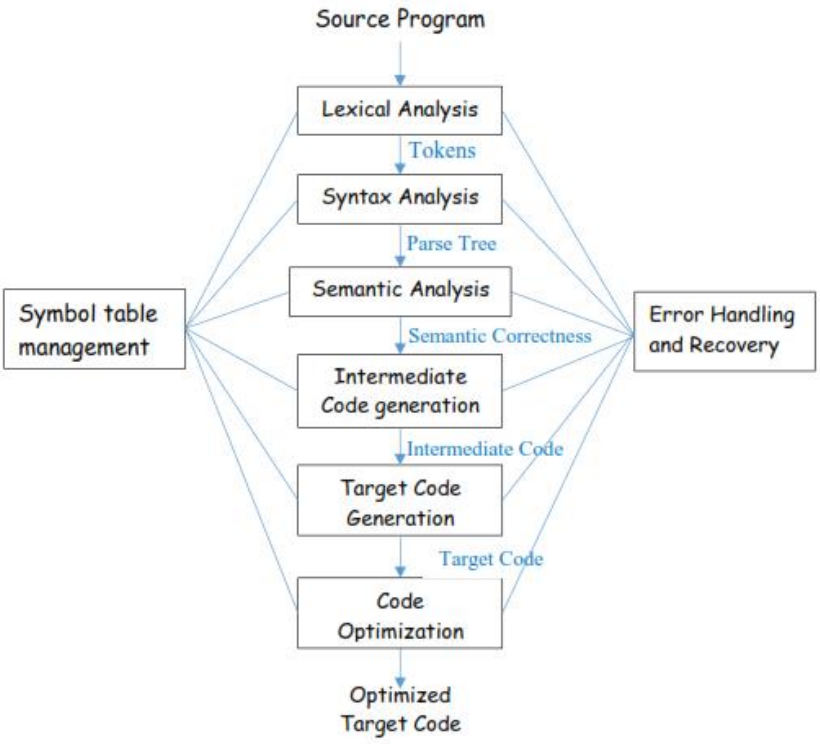
[5] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 29

2.0 Phases of a Compiler

In order to ease the process of development and understanding, a compiler can conceptually be divided into the following phases:

i) *Lexical analysis*
ii) *Syntax analysis*
iii) *Semantic analysis*
iv) *Intermediate code generation*
v) *Target code generation*
vi) *Code optimization*
vii) *Symbol table management*
viii) *Error handling and recovery*[6]

The following diagram summarizes the relationship between these modules[7]:



[6] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 30

[7] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 31, 32

I. **Lexical Analysis**

- ✓ This is the initial part of reading and analyzing the program text
- ✓ The text is read and divided into **tokens**, each of which corresponds to a **symbol** in the programming language e.g. a variable name, number, keyword, etc.
- ✓ The idea is to basically identify **valid words in the input source program[8].**

II. **Syntax Analysis**

- ✓ This phase takes the list of tokens produced by the lexical analysis and **arranges them into a tree structure (syntax tree**) that reflects the structure of the program.
- ✓ It is generally used to establish if the program is grammatically correct.
- ✓ This phase is often called **parsing[9].**

III. **Semantic Analysis**

- ✓ This phase analyses the syntax tree to determine if the program **violates certain consistency requirements[10]**

  Examples: -

  - ✓ Is there a variable used without being declared?
  - ✓ Has a variable been used in a context that doesn't make sense given the type of variable such as trying to assign a value greater than the variable?

IV. **Intermediate Code Generation**

- ✓ This phase is used to translate the program into a simple machine independent intermediate language.
- ✓ This process helps to re-target the compiler generating code from one processor to another[11].

[8] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 30

[9] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 33

[10] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 33

[11] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 34

Generally, generating machine code directly from source code entails two problems

i) With ***m***-languages and ***n***-target machines, we need to write ***m*** × ***n*** compilers[12]

ii) The code optimizer which is one of the largest and very-difficult-to-write components of any compiler cannot be re-used

- ✓ A machine-independent code optimizer can be written by converting source code to an intermediate code

V. Target Code Generation

- ✓ This is used for template substitution i.e. for every statement of the intermediate language, a predefined target language template is used to generate the final code[13].
- ✓ Basically, it is used to translate the intermediate language to specific machine architecture.

VI. Code Optimization

- ✓ This phase is used to make the code efficient.
- ✓ It is generally used to eliminate redundant codes generated during the process of compilation[14].

VII. Symbol Table Management

- ✓ A symbol table is a data structure that holds information about all symbols held in the source program[15].
- ✓ Information stored includes variable names, their size, types, etc.
- ✓ It does not form part of the final code generated.

VIII. Error Handling and Recovery

- ✓ Error handling is used to give an indication regarding the type of error.

---

[12] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

[13] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 35

[14] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 35

[15] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 36

- ✓ It should be detailed enough to pin- point the error and at the same time should not be too wordy to confuse the user.
- ✓ Recovery is used to undo some of the processing already carried by the parser so as to reach a state where it can proceed again i.e. prevents parser from coming out on the first error it encounters[16].

3.0 Compiler Construction Tools

Compilers operate under modern general software development environments containing tools such as language editors, debuggers, version managers, profilers, test harnesses etc. In order to enhance the implementation of various compiler phases, other more specialized tools have been created in addition to the general software development environment. These tools include: -

1. ***Parser generators*** which produce syntax analyzers automatically from a grammatical description of a programming language.
2. ***Scanner generators*** which produce lexical analyzers from regular-expressions of the language tokens.
3. ***Syntax-directed translation engines*** that produce collections of routines to generate parse trees and intermediate code.
4. ***Code-generator generators*** that produce a code generator from a collection of rules which is used to translate each operation of the intermediate language into the machine language equivalent for a target machine.
5. ***Data-flow analysis engines*** that facilitate the gathering of information about how values are transmitted within the program sections. Data-flow analysis is an important part of code optimization.
6. ***Compiler-construction toolkits*** which provide an integrated set of routines useful when constructing various compiler phases[17].

[16] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 35

[17] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 37

Four Qualities of a Good Compiler[18]

i) **Correct:** the meaning of sentences must be preserved

ii) **Robust:** wrong input is the common case

- ✓ Compilers and interpreters can't just crash on wrong input
- ✓ They need to diagnose all kinds of errors safely and reliably

iii) **Efficient:** resource usage should be minimal in two ways

a) The process of compilation or interpretation itself is efficient

b) The generated code is efficient when interpreted

iv) **Usable:** integrate with environment, accurate feedback

- ✓ Work well with other tools (editors, linkers, debuggers...)
- ✓ Descriptive error messages, relating accurately to source

Example of a Compilation Process

Consider the following program[19]:

```
program
        var x1,x2: integer; {integer variable declaration}
        var xR: real;     {real variable declaration
begin
        xR:=x1+x2*10;     {assignment statement}
end.
```

[1].**Lexical Analysis**

The lexical analyzer scans the program looking for major lexical elements (**tokens**).

It produces the following[20]:

program, var, x1, ',', x2, ':', integer, ';', var, xR, ':', real, ';', begin, xR '=', x1, '+', x2, '*', 10,';', end, '.'

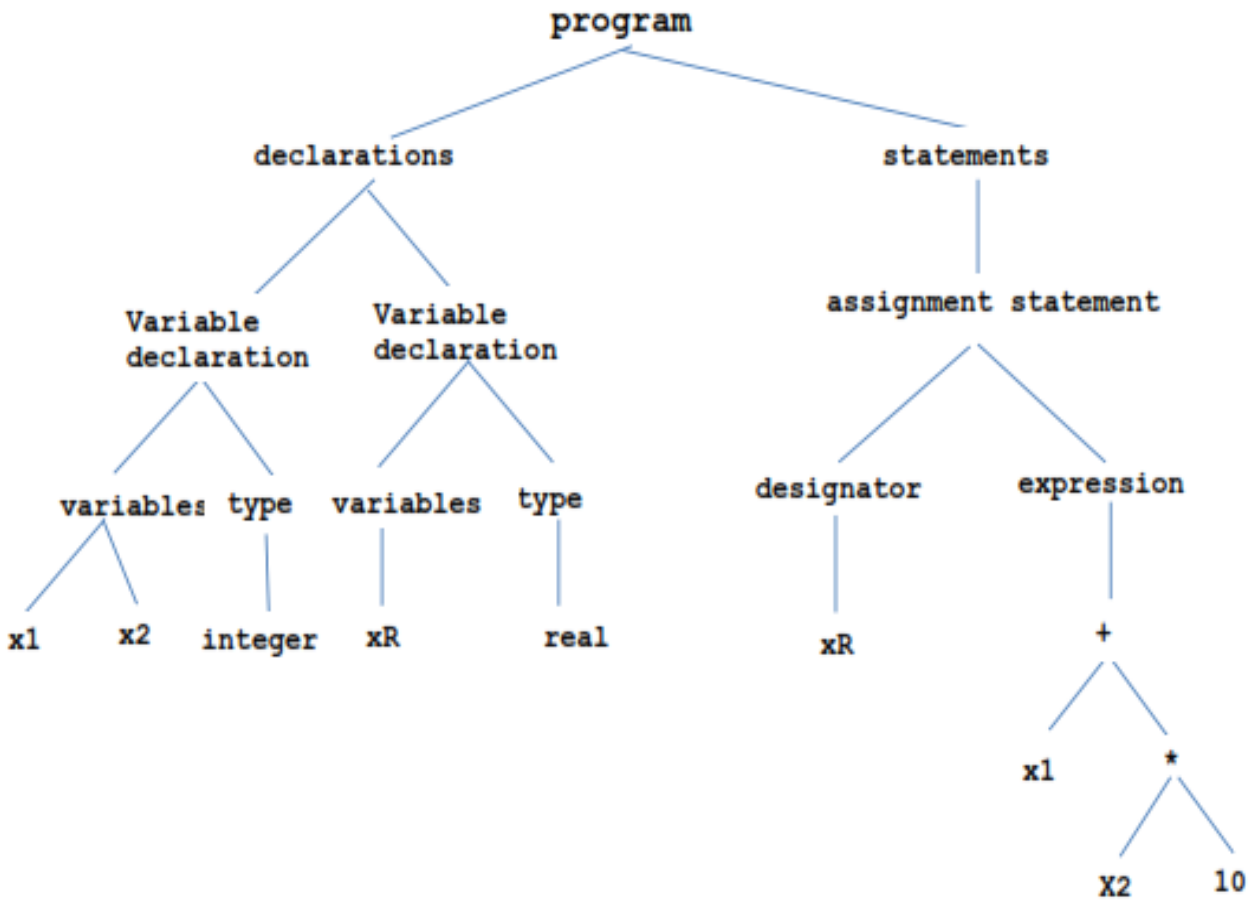
[18] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 40

[19] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 32

[20] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

[2]. **Syntax Analysis**

- ✓ It analyses for grammatical correctness and for a grammatically correct program.
- ✓ It produces the following parse tree[21]:



[3]. **Code Generation**

- ✓ It transverses the parse tree and generates code for the input program.
- ✓ Basically, it assumes a stack-oriented machine with instructions such as PUSH, ADD, MULT, STORE.
- ✓ The following is a sample code for the program:

```
PUSH x2
PUSH 10
MULT
PUSH x1
```

[21] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 33

ADD

PUSH @ xR

STORE

Note: @returns the address of the variable following it.[22]

[4].**Symbol Table** is a table created by the lexer and parser and later used by the code generator[23]. The following is a sample symbol table generated.

Name	Class	Type
X1	Variable	Integer
X2	Variable	Integer
xR	Variable	Real

[22] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 33

[23] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 36

Content Covered in Week 1: Introduction to Compiler Construction

At the end of the lecture, we were able to:

(i) Define a compiler, interpreter and translator

(ii) Describe the phases of a compiler

(iii) Explain the qualities of a good compiler.

Course Text Books

1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; *Addison-Wesley Pub Co*, ISBN: 0201100886 (2007))
2. Compiler Construction: Principles and practices; Kenneth C. Louden; *Cengage Learning*; 1st edition, ISBN-10 : 0534939724 (1997)
3. Basics of Compiler Design: Torben Mogensen; *DIKU University of Copenhagen Universitetsparken 1 DK-2100 Copenhagen DENMARK* (2007)
4. Engineering a Compiler: 2nd edition; Keith D. Cooper and Linda Torczon; *Morgan Kaufmann Publishers* ISBN: 978-0-12-088478-0 (2003)
5. Compiler Design: Santanu Chattopadhyay; *PHI Learning Publishers*, ISBN 812032725X, 9788120327252 (2005)