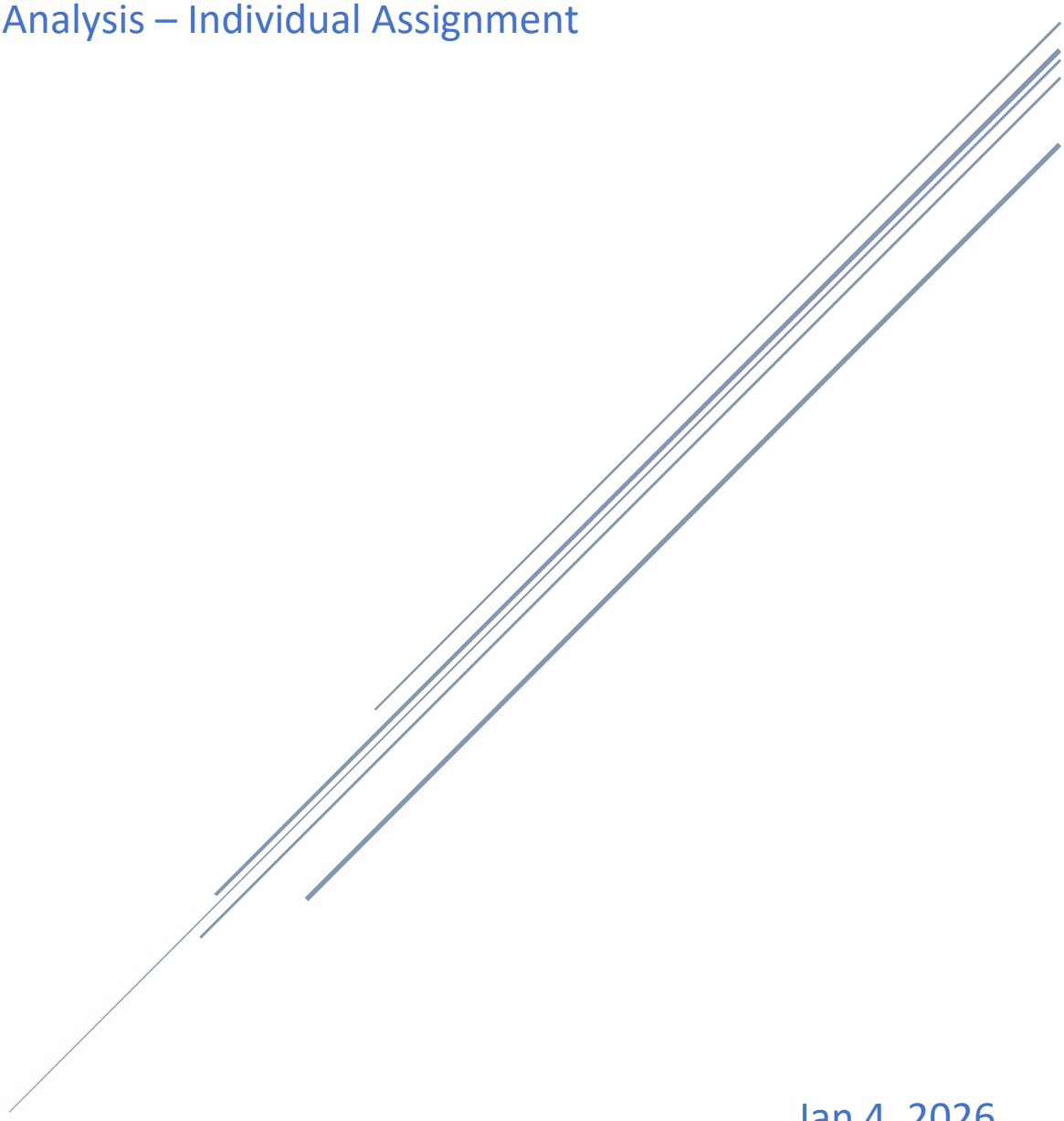
Bahir Dar University
Institute of Technology /BiT

Principles of Compiler Design

Syntax Analysis – Individual Assignment



Jan 4, 2026

Yonatan Ayisheshim [BDU1508377]

Table of Contents

Introduction

Syntax analysis is a critical phase of compiler design that verifies whether a sequence of tokens generated by the lexical analyzer conforms to the grammatical structure of the programming language. This assignment focuses on understanding syntactic structures through theory, practical C++ implementation, and grammar-based problem solving.

Theory - Difference between Parse Tree and Abstract Syntax Tree (AST)

1.1 Parse Tree

A **parse tree** (also known as a concrete syntax tree) is a hierarchical representation of the syntactic structure of a string derived according to the grammar rules of a language.

Characteristics:

- Represents **every grammar symbol** (terminals and non-terminals)
- Closely follows the **context-free grammar (CFG)**
- Useful for **syntax verification and teaching**
- Often **large and verbose**

Example: For grammar rule,

```
E → E + T
```

The parse tree explicitly shows `E`, `+`, and `T` as nodes.

1.2 Abstract Syntax Tree (AST)

An **abstract syntax tree (AST)** is a condensed, semantic-oriented representation of the source code.

Characteristics:

- Omits unnecessary grammar symbols (parentheses, punctuation)
- Captures **essential hierarchical structure**
- More **compact and efficient**
- Used in **semantic analysis, optimization, and code generation**

Example: The expression,

```
a + b
```

In an AST is represented simply as:

```
    +
   / \
  a   b
```

1.3 Key Differences

Aspect	Parse Tree	Abstract Syntax Tree (AST)
Grammar dependency	Strictly grammar-based	Grammar-independent
Size	Large	Compact
Purpose	Syntax validation	Semantic analysis
Includes terminals	Yes	No
Used in	Parsing phase	Later compiler phases

C++ Implementation - Checking Balanced Square Brackets `[]`

2.1 Problem Statement

Write a C++ program that checks whether a given string contains **balanced square brackets** (`[` and `]`).

2.2 Approach

A stack-based approach is used:

- Push `'['` when encountered
- Pop when `']'` is encountered
- If a closing bracket appears with an empty stack → unbalanced
- Stack must be empty at the end for the string to be balanced

2.3 C++ Program

The C++ program is attached separately for execution convenience.

2.4 Example Execution

Input	Output
`[[]]`	Balanced
`[][[]]`	Balanced
`[]]`	Not Balanced
`[[[]`	Not Balanced

Conclusion

This assignment demonstrated:

- The conceptual difference between **parse trees and ASTs**
- Practical application of **stack-based parsing** in C++

References

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. *Compilers: Principles, Techniques, and Tools* (2nd ed.)
- Compiler Design Lecture Notes