BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

COMPUTING FACULTY

DEPARTMENT OF COMPUTER SCIENCE

Course Title: Compiler Design(CoSc4022)

Individual Assignment

Topic: Syntax Analysis individual assignment 01

Student Name: Adem Seidu

ID Number:1411142

Submitted to:Mr.Wendimu B

Submission date: 01/05/2015EC

Introduction

Syntax analysis is a fundamental phase in the process of compiler design. It is responsible for analyzing the structure of source code to ensure that it conforms to the grammatical rules of a programming language. By using formal grammars and parsing techniques, syntax analysis helps detect structural errors and organizes tokens into meaningful constructs.

This assignment focuses on understanding the core concepts of syntax analysis, including different parsing techniques such as LL(1) and LR(0) parsers. It also emphasizes practical applications of grammar analysis through parse tree construction and the implementation of compiler-related programs using C++. Through theoretical explanation, problem-solving, and programming tasks, this assignment aims to strengthen the learner's understanding of how syntax analysis bridges lexical analysis and later phases of the compiler such as semantic analysis and code generation.

Question 1: Theory

Explain the Difference Between LL(1) and LR(0) Parsers

Syntax analysis is the phase of a compiler that checks whether the source code follows the grammatical structure of the programming language. Two important parsing techniques used in syntax analysis are **LL(1)** and **LR(0)** parsers.

LL(1) Parser

An **LL(1) parser** is a **top-down parser** that reads the input from **Left to right** and produces a **Leftmost derivation** using **1 lookahead symbol**.

Characteristics of LL(1) Parser:

- Top-down parsing technique
- Uses recursive descent or predictive parsing
- Requires grammar to be free from left recursion
- Uses FIRST and FOLLOW sets
- Easy to implement
- Detects errors early but has limited grammar power

LR(0) Parser

An **LR(0) parser** is a **bottom-up parser** that reads the input from **Left to right** and produces a **Rightmost derivation in reverse** with **no lookahead symbol**.

Characteristics of LR(0) Parser:

- Bottom-up parsing technique

- Uses shift-reduce parsing
- Handles a larger class of grammars
- More powerful than LL(1)
- Uses parsing states and parsing tables
- Implementation is more complex

Comparison Table

Feature	LL(1) Parser	LR(0) Parser
Parsing Method	Top-down	Bottom-up
Derivation	Leftmost	Rightmost (reverse)
Lookahead	1 symbol	No lookahead
Grammar Power	Limited	More powerful
Left Recursion	Not allowed	Allowed
Implementation	Simple	Complex
Error Detection	Early	Slightly delayed

Question 2: C++ Programming

Write a C++ Program to Remove Single-Line Comments (//)

The following C++ program removes all single-line comments (//) from a source code file while preserving the original code structure.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
    ifstream inputFile("input.cpp");
    ofstream outputFile("output.cpp");

    if (!inputFile.is_open()) {
        cout << "Error opening input file." << endl;
        return 1;
    }

    string line;
    while (getline(inputFile, line)) {
        size_t commentPos = line.find("//");
        if (commentPos != string::npos) {
            line = line.substr(0, commentPos);
        }
        outputFile << line << endl;
```

```
    }

    inputFile.close();
    outputFile.close();

    cout << "Comments removed successfully." << endl;
    return 0;
}
```

Explanation:

- The program reads the source file line by line.
- It checks for the presence of //.
- If found, everything after // is removed.
- The cleaned code is written to an output file.

Question 3: Problem Solving

Grammar Given

S → aSbS | ε

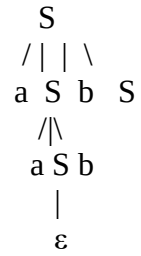
Input String

aabb

Derivation

1. S → aSbS
2. → a a S b S b S
3. → a a ε b ε b ε
4. → aabb

Parse Tree for "aabb"



Explanation

- The grammar allows recursive nesting using aSbS.
- The empty string (ε) ends recursion.
- The input string aabb is generated by expanding S twice and then applying ε-productions.

- The parse tree shows how terminals and non-terminals derive the string step by step.

Conclusion

This assignment demonstrated key concepts of syntax analysis including LL(1) and LR(0) parsing, grammar-based problem solving, and practical implementation using C++. Understanding these concepts provides a strong foundation for advanced compiler phases such as semantic analysis and code generation.