BAHIR DAR UNIVERSITY

Bahir Dar Institute of Technology Faculty of Computing

Principles of Compiler Design

Kalkidan Mehiret …………….. BDU1506674

Compiler Design Assignments

ASSIGNMENT 1

Part A: **Difference between LL(1) and LR(1) Parsers**

Meaning

LL(1) Parser:

- Left to right input scanning
- Leftmost derivation
- Top-down parsing
- Cannot handle left recursion
- Simple implementation

LR(1) Parser:

- Left to right input scanning
- Rightmost derivation in reverse
- Bottom-up parsing
- Can handle left recursion
- More powerful and complex

Feature	**LL(1) Parser**	**LR(1) Parser**
Parsing approach	Top-down	Bottom-up
Derivation	Leftmost	Rightmost (reverse)
Grammar power	Less powerful	More powerful
Left recursion	Not allowed	Allowed
Implementation	Simple	Complex
Error detection	Less effective	Better

LL(1) parsers are simple and easy to implement, while LR(1) parsers can handle more complex grammars and provide better error detection.

Part B: C++ Program to Check Palindrome

A palindrome is a string that reads the same forward and backward. Example: `madam`.
Algorithm:

1. Read the input string
2. Reverse the string
3. Compare original and reversed strings
4. Print the result

C++ Program:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
   string str, rev = "";
   cin >> str;
   for (int i = str.length() - 1; i >= 0; i--)
      rev += str[i];
   if (str == rev)
      cout << "Palindrome";
   else
      cout << "Not Palindrome";
   return 0;
}
```

Part C: Parse Tree

Given Grammar:
$S \rightarrow aSb \mid \varepsilon$

Given String: aaabbb

Step 1: Derivation

S

⇒ aSb

⇒ a(aSb)b

⇒ a(a(aSb)b)b

⇒ a(a(aεb)b)b

Step 2: Parse Tree

Parse Tree:

```
        S
   /    |   \
  a     S   b
      / | \
    a   S   b
       /|\
      a S b
        |
        ε
```