UNIVERSITI TEKNOLOGI MARA
Cawangan Terengganu

COURSE : PRINCIPLES OF COMPILERS

COURSE CODE : CSC569

ASSESSMENT : MINI PROJECT

SUBMISSION : 9 JULY 2026 (BEFORE 11.59 P.M.)

No.	Student Name	Student No.
1.	MUHAMMAD AIMAN BIN JAMALUDDIN	2025168045
2.	NAJWA IZZIATI BINTI HASSANUDDIN	2024963145

Group: CDCS2304B1

PREPARED FOR: DR NORLINA BINTI MOHD SABRI

Table of Contents

Table of Contents 2

1. The New Grammar 3

2. How to Get the Selection Set 3

Step 1 – Nullable Rules ..... **Error! Bookmark not defined.**

Step 2 – Nullable Non-terminals ..... **Error! Bookmark not defined.**

Step 3 – BDW (Begins Directly With) ..... **Error! Bookmark not defined.**

Step 4 – BW (Begins With) ..... **Error! Bookmark not defined.**

Step 5 – First(X) for Non-terminals ..... **Error! Bookmark not defined.**

Step 6 – First Set of Each Rule ..... **Error! Bookmark not defined.**

Step 7 – DEO (Direct End Of) ..... **Error! Bookmark not defined.**

Step 8 – EO (End Of) ..... **Error! Bookmark not defined.**

Step 9 – FDB (Followed Directly By) ..... **Error! Bookmark not defined.**

Step 10 – FB (Followed By, via EO) ..... **Error! Bookmark not defined.**

Step 11 – FB from Start Symbol (↵, end of input) ..... **Error! Bookmark not defined.**

Step 12 – Selection Sets ..... **Error! Bookmark not defined.**

3. Recursive Descent Parser — Program Code 6

4. Sample Input and Output 11

Sample 1 – Accepted 11

Sample 2 – Accepted (all four operators + nested parentheses) 12

Sample 3 – Rejected 13

1. The New Grammar

The base grammar used for this project is G17 (Translation Grammar, page 136 of the textbook). Two additional rules were added — one for subtraction (–) and one for division (/) — so that the grammar now supports all four arithmetic operators (+, –, *, /) and contains exactly 10 rules, as required by the assignment.

As an extension, the grammar also recognises Arabic-script equivalents for each terminal symbol (var/num → ,(غ → (,ق → / ,ض → * ,ط → - ,و → + ,ب so the parser can accept either Latin operators/digits or their Arabic-letter counterparts.

No.	Rule
1	Expr → Term Etail
2	Etail → + Term Etail
3	Etail → - Term Etail (added rule — subtraction)
4	Etail → ε
5	Term → Factor Ttail
6	Ttail → * Factor Ttail
7	Ttail → / Factor Ttail (added rule — division)
8	Ttail → ε
9	Factor → (Expr)
10	Factor → var | num | ب

Rules 3 and 7 are the two new rules added to the original G17 grammar, giving complete coverage of add, minus, multiply and divide as required by the project brief.

2. How to Get the Selection Set

The Selection Set for each rule was derived by working through the standard First/Follow procedure used in class: Nullable rules and non-terminals, BDW/BW (Begins-With) relations, First sets, DEO/EO (End-Of) relations, FDB/FB (Followed-By) relations, and finally the Selection Set of every rule.

Step 1 – Nullable Rules

Rule 4 (Etail → ε) and Rule 8 (Ttail → ε) are nullable.

Step 2 – Nullable Non-terminals

Etail and Ttail are nullable non-terminals.

Step 3 – BDW (Begins Directly With)

Expr BDW Term
Etail BDW +, -
Term BDW Factor
Ttail BDW *, /
Factor BDW (, var, num, ب

Step 4 – BW (Begins With)

Direct: Expr BW Term, Etail BW +, Etail BW -, Term BW Factor, Ttail BW *, Ttail BW /, Factor BW (, Factor BW var, Factor BW num, Factor BW ب.

Transitive (through Term BW Factor and Expr BW Term): Term BW (, Term BW var, Term BW num, Term BW ب; Expr BW Factor, Expr BW (, Expr BW var, Expr BW num, Expr BW ب.

Reflexive: every symbol begins with itself (Expr BW Expr, Etail BW Etail, Term BW Term, Ttail BW Ttail, Factor BW Factor, and each terminal BW itself).

Step 5 – First(X) for Non-terminals

First(Expr) = { (, var, num, ب }
First(Etail) = { +, - }
First(Term) = { (, var, num, ب }
First(Ttail) = { *, / }
First(Factor) = { (, var, num, ب }

Step 6 – First Set of Each Rule

First(1) = { (, var, num, ب }
First(2) = { + }
First(3) = { - }
First(4) = { } (nullable)
First(5) = { (, var, num, ب }
First(6) = { * }
First(7) = { / }
First(8) = { } (nullable)
First(9) = { (}
First(10) = { var, num, ب }

Step 7 – DEO (Direct End Of)

Etail DEO Expr
Ttail DEO Term
Term DEO Expr (rightmost Etail is nullable)
Factor DEO Term (rightmost Ttail is nullable)
), var, num, ب DEO Factor

Step 8 – EO (End Of)

From DEO and transitivity: Etail EO Expr, Ttail EO Term, Term EO Expr, Factor EO Term, and Factor EO Expr (since Factor EO Term and Term EO Expr). Also), var, num, ب EO Factor, EO Term and EO Expr. Reflexively, Expr EO Expr, Etail EO Etail, Term EO Term, Ttail EO Ttail, Factor EO Factor.

Step 9 – FDB (Followed Directly By)

Term FDB Etail → Term FDB +, Term FDB -
+ FDB Term → + FDB (, var, num, - ; ب FDB Term → - FDB (, var, num, ب
Factor FDB Ttail → Factor FDB *, Factor FDB /
* FDB Factor → * FDB (, var, num, / ; ب FDB Factor → / FDB (, var, num, ب
(FDB Expr → (FDB (, var, num, ب
Expr FDB)

Step 10 – FB (Followed By, via EO)

Term EO Expr, Expr FDB) → Term FB)
Factor EO Term, Term FDB +/- → Factor FB +, Factor FB -
Factor EO Term, Term FB) → Factor FB)
Etail EO Expr, Expr FDB) → Etail FB)
Ttail EO Term, Term FDB +/-, Term FB) → Ttail FB +, Ttail FB -, Ttail FB)

Step 11 – FB from Start Symbol (↲, end of input)

Expr FB ↲
Etail EO Expr → Etail FB ↲
Term EO Expr → Term FB ↲
Ttail EO Term → Ttail FB ↲
Factor EO Expr → Factor FB ↲

Step 12 – Selection Sets

Sel(1) = { (, var, num, ب }
Sel(2) = { + }
Sel(3) = { - }
Sel(4) = Fol(Etail) = {), ↲ }
Sel(5) = { (, var, num, ب }
Sel(6) = { * }
Sel(7) = { / }
Sel(8) = Fol(Ttail) = { +, -,), ↲ }
Sel(9) = { (}
Sel(10) = { var, num, ب }

Because none of the rules that share a left-hand side (Etail: rules 2, 3, 4; Ttail: rules 6, 7, 8; Factor: rules 9, 10) have overlapping Selection Sets, the grammar is confirmed to be LL(1) and can be parsed with a single-token-lookahead recursive descent parser which is exactly what was implemented below.

3. Recursive Descent Parser — Program Code

The program is a Java Swing desktop application. The recursive descent functions (Expr, Etail, Term, Ttail, Factor) map directly onto the 10 grammar rules above; each function reads one lookahead character (inp) and decides which rule to apply based on the Selection Sets derived in Section 2. Invalid input triggers a rejected exception with the specific reason, which is caught and displayed in the Output field; a full pass with no leftover character's triggers Accept.

```
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.*;


public class ParserProject extends JFrame {

    // UI stuff
    private JTextField input_field;
    private JTextField resultBox;
    private JTextField output_txt;

    // variables from textbook pg 104 - trying to match the RDP structure
    char inp;
    String inputStr;
    int pos;

    public ParserProject() {
        // setup window...
        setTitle("CSC569 - Recursive Descent Parser (Group Project)");
        setSize(550, 350);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());

        // banana color theme (Najwa picked these colors)
        Color bright_yellow = new Color(255, 225, 53);
        Color paleYellow = new Color(255, 253, 220);
        Color darkBrown = new Color(77, 51, 25);

        Font titleFont = new Font("SansSerif", Font.BOLD, 22);
        Font label_font = new Font("SansSerif", Font.BOLD, 14);
        Font textFont = new Font("Monospaced", Font.PLAIN, 16);

        // header
        JPanel header = new JPanel();
        header.setBackground(bright_yellow);
        JLabel title_label = new JLabel("Compiler Parser Engine");
        title_label.setForeground(darkBrown);
        title_label.setFont(titleFont);
        header.setBorder(new EmptyBorder(15, 10, 15, 10));
        add(header, BorderLayout.NORTH);

        // main panel
        JPanel center_panel = new JPanel(new GridBagLayout());
        center_panel.setBackground(paleYellow);
        center_panel.setBorder(new EmptyBorder(20, 30, 20, 30));

        GridBagConstraints gbc = new GridBagConstraints();
```

```
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.insets = new Insets(10, 10, 10, 10);

// input field
gbc.gridx = 0; gbc.gridy = 0; gbc.weightx = 0.2;
JLabel inputLbl = new JLabel("Enter Input:");
inputLbl.setFont(label_font);
inputLbl.setForeground(darkBrown);
center_panel.add(inputLbl, gbc);

gbc.gridx = 1; gbc.gridy = 0; gbc.weightx = 0.8;
input_field = new JTextField();
input_field.setFont(textFont);
input_field.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(darkBrown, 1),
    BorderFactory.createEmptyBorder(5, 5, 5, 5)));
center_panel.add(input_field, gbc);

// result box
gbc.gridx = 0; gbc.gridy = 1; gbc.weightx = 0.2;
JLabel resultLbl = new JLabel("Result:");
resultLbl.setFont(label_font);
resultLbl.setForeground(darkBrown);
center_panel.add(resultLbl, gbc);

gbc.gridx = 1; gbc.gridy = 1; gbc.weightx = 0.8;
resultBox = new JTextField();
resultBox.setFont(textFont);
resultBox.setEditable(false);
resultBox.setBackground(Color.WHITE);
resultBox.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(darkBrown, 1),
    BorderFactory.createEmptyBorder(5, 5, 5, 5)));
center_panel.add(resultBox, gbc);

// error messages here
gbc.gridx = 0; gbc.gridy = 2; gbc.weightx = 0.2;
JLabel outLbl = new JLabel("Output:");
outLbl.setFont(label_font);
outLbl.setForeground(darkBrown);
center_panel.add(outLbl, gbc);

gbc.gridx = 1; gbc.gridy = 3; gbc.gridwidth = 2; gbc.weightx = 1.0;
output_txt = new JTextField();
output_txt.setFont(textFont);
output_txt.setEditable(false);
output_txt.setBackground(Color.WHITE);
output_txt.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(darkBrown, 1),
    BorderFactory.createEmptyBorder(5, 5, 5, 5)));

gbc.gridx = 0; gbc.gridy = 3; gbc.gridwidth = 2;
center_panel.add(output_txt, gbc);

add(center_panel, BorderLayout.CENTER);

// button at the bottom
JPanel bottom_btn_panel = new JPanel(new FlowLayout(FlowLayout.LEFT));
bottom_btn_panel.setBackground(paleYellow);
bottom_btn_panel.setBorder(new EmptyBorder(0, 15, 20, 0));

JButton parse_button = new JButton("Parse String");
parse_button.setFont(new Font("SansSerif", Font.BOLD, 14));
parse_button.setBackground(darkBrown);
parse_button.setForeground(bright_yellow);
parse_button.setFocusPainted(false);
parse_button.setContentAreaFilled(false);
parse_button.setOpaque(true);
parse_button.setPreferredSize(new Dimension(150, 40));
```

```
    bottom_btn_panel.add(parse_button);
    add(bottom_btn_panel, BorderLayout.SOUTH);

    parse_button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // calling the main logic
            parse(input_field.getText());
        }
    });
}

// --- recursive logic for the parser ---

void parse(String text) {
    resultBox.setForeground(Color.BLACK);
    output_txt.setForeground(Color.BLACK);

    // trim whitespace - dr said this makes it easier
    inputStr = text.replaceAll("\\s+", "");
    pos = -1;

    try {
        inp = getInp();
        Expr();

        // check for end of string, '$' is the marker
        if (inp == '$') {
            accept();
        } else {
            reject("Extra characters found at the end of input.");
        }
    } catch (Exception ex) {
        // shows the error message in the GUI instead of just console
        resultBox.setText("Reject");
        resultBox.setForeground(Color.RED);
        output_txt.setText(ex.getMessage());
        output_txt.setForeground(Color.RED);
    }
}

void accept() {
    resultBox.setText("Accept");
    resultBox.setForeground(new Color(40, 167, 69));
    output_txt.setText("Input successfully parsed.");
}

void reject(String reason) throws Exception {
    throw new Exception("Error: " + reason);
}

char getInp() {
    pos++;
    if (pos < inputStr.length()) {
        return inputStr.charAt(pos);
    }
    return '$';
}

boolean isArabicLetter(char c) {
    // range for arabic chars - found this online
    return c >= '\u0621' && c <= '\u064A';
}

// --- Grammar rules section ---

void Expr() throws Exception {
    if (inp == '(' || isArabicLetter(inp) || Character.isDigit(inp)) {
        Term();
        Etail();
    } else {
```

```
        reject("Expr expected '(', Arabic Letter, or Number, got '" + inp + "'");
    }
}

void Etail() throws Exception {
    // plus or arabic 'waw'
    if (inp == '+' || inp == 'و') {
        inp = getInp();
        Term();
        Etail();
    }
    // minus or arabic 'tha'
    else if (inp == '-' || inp == 'ط') {
        inp = getInp();
        Term();
        Etail();
    }
    else if (inp == ')' || inp == 'غ' || inp == '$') {
        // empty/epsilon
    }
    else {
        reject("Etail expected '+', '-', 'ط', 'و or end, got '" + inp + "'");
    }
}

void Term() throws Exception {
    if (inp == '(' || isArabicLetter(inp) || Character.isDigit(inp)) {
        Factor();
        Ttail();
    } else {
        reject("Term expected '(', Arabic Letter, or Number, got '" + inp + "'");
    }
}

void Ttail() throws Exception {
    // mult or arabic 'daad'
    if (inp == '*' || inp == 'ض') {
        inp = getInp();
        Factor();
        Ttail();
    }
    // div or arabic 'qaf'
    else if (inp == '/' || inp == 'ق') {
        inp = getInp();
        Factor();
        Ttail();
    }
    else if (inp == '+' || inp == 'و' || inp == '-' || inp == ' || 'ط
            inp == ')' || inp == 'غ' || inp == '$') {
        // epsilon
    }
    else {
        reject("Ttail expected '*', '/', 'ق', 'ض or follow symbols, got '" + inp + "'");
    }
}

void Factor() throws Exception {
    if (inp == '(') {
        inp = getInp();
        Expr();
        if (inp == ')') {
            inp = getInp();
        } else {
            reject("Factor expected closing ')', got '" + inp + "'");
        }
    }
    else if (isArabicLetter(inp) || Character.isDigit(inp)) {
        // rule 10 handles variables and numbers
        inp = getInp();
    }
    else {
        reject("Factor expected '(', Arabic Letter, or Number, got '" + inp + "'");
    }
```

```
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
                } catch (Exception e) {}

                ParserProject myProject = new ParserProject();
                myProject.setVisible(true);
            }
        });
    }
}
```

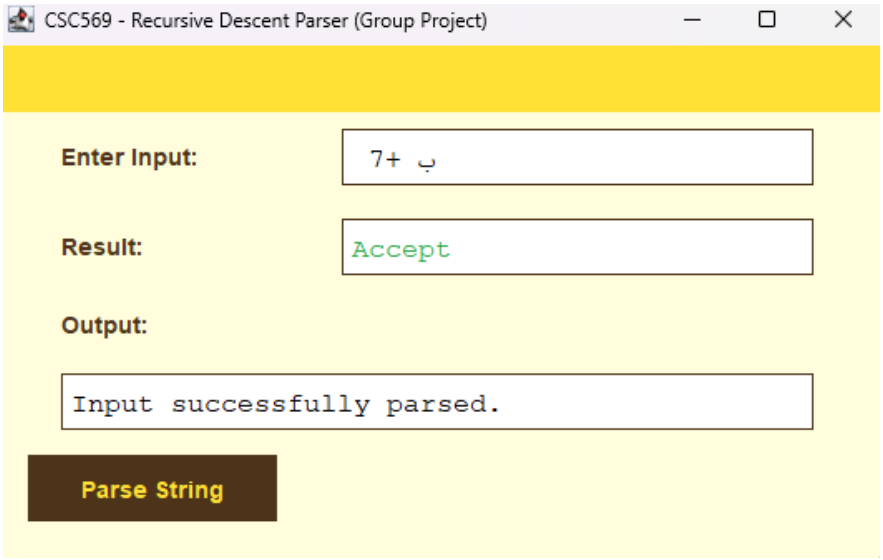
4. Sample Input and Output

Three representative test runs are shown below: two accepted strings (covering all four operators and nested parentheses) and one rejected string, together with the exact error message produced by the parser.

Sample 1 – Accepted

Enter Input	7 ب
Result	**Accept**
Output	Input successfully parsed.

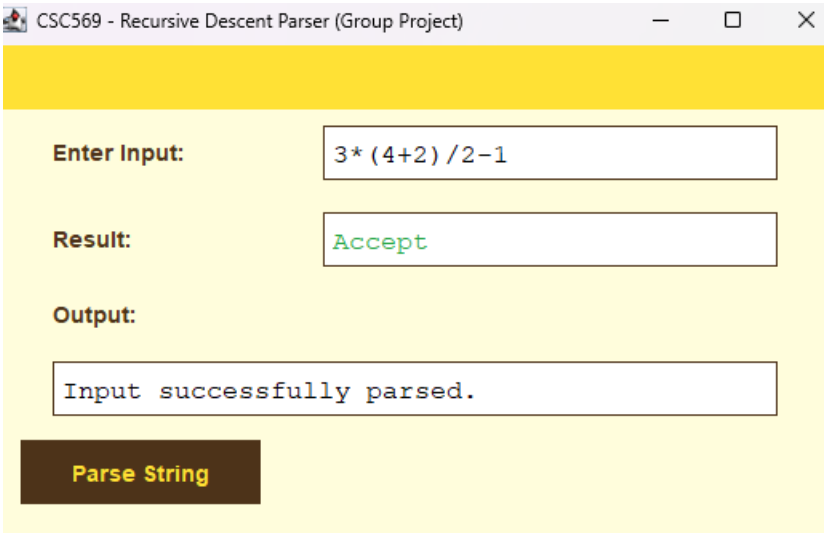
Trace: Factor reads '7' (Rule 10); Ttail sees '+' and takes the ε-rule (Rule 8, since + is in Sel(8)); Etail sees '+' (Rule 2), consumes it, then parses Term → Factor ', 'ب so the whole string is consumed with no characters left over → Accept.



Sample 2 – Accepted (all four operators + nested parentheses)

Enter Input	3*(4+2)/2-1
Result	**Accept**
Output	Input successfully parsed.

Trace: Factor reads '3', Ttail sees '*' (Rule 6) and calls Factor again, which sees '(' (Rule 9) and recursively parses the nested expression '4+2' before consuming the matching ')'. Ttail then sees '/' (Rule 7) and consumes Factor '2'; back at the Etail level, '-' is matched (Rule 3) and Factor '1' is consumed. All four operators (+, -, *, /) and one level of nested parentheses are exercised, and the string is fully consumed → Accept.



Sample 3 – Rejected

Enter Input	5++3
Result	**Reject**
Output	Error: Term expected '(', Arabic Letter, or Number, got '+'

Trace: Factor reads '5', Ttail reduces by ε (Rule 8), Etail matches the first '+' (Rule 2) and calls Term to look for an operand but the very next character is a second '+', which is not in Sel(1)/Sel(5) (not '(', a digit, or an Arabic letter). Term() therefore throws the rejection with that exact reason, which the program catches and displays in the Output field in red.

