ROLEPLAY ACTIVITY REPORT

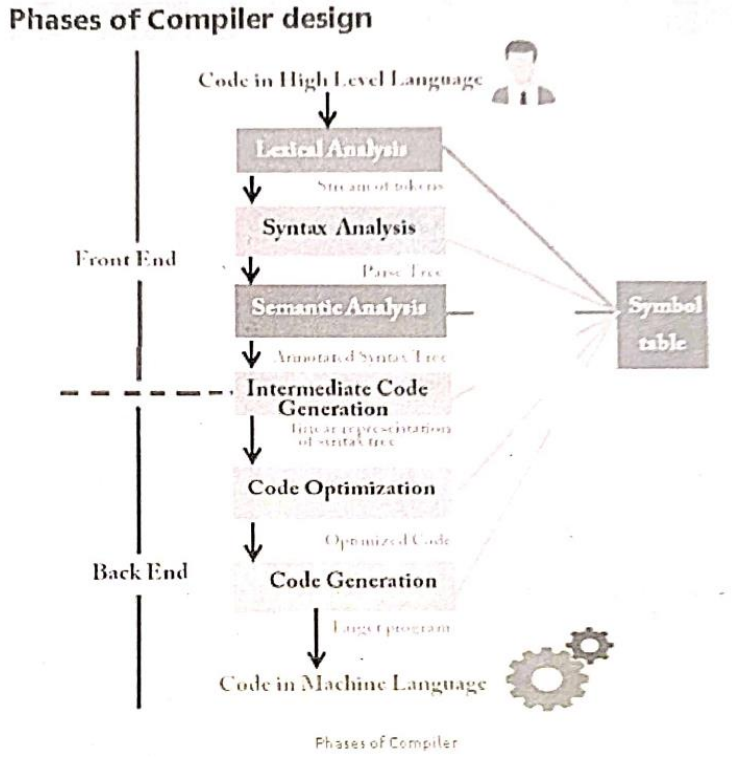
Topic-Phases of a Compiler

Department of Computer Science and Engineering

Role Play on: ***Phases of a Compiler***
Venue: Room No. 410
Date: [22/03/2025]

ROLE –PLAY ACTIVITY (COMPILER-DESIGN)

PHASES OF COMPILER



Phases of Compiler

OBJECTIVE OF THE ACTIVITY:

The most common teaching method adopted to deliver the working of various phases of the compiler is the lecture-based method, where the instructor explains the concepts of each phase using the blackboard or presentations. While effective, this method may not fully engage all types of learners.

To make the learning process more interactive and student-centric, a **Role Play Activity** was organized on the topic **"Phases of a Compiler."** This activity aimed to break down the complex functionality of compiler phases into simple, enactable parts, thereby improving conceptual understanding and participation.

Activity Description:

Each compiler phase — Lexical Analysis, Syntax Analysis, Semantic Analysis, Intermediate Code Generation, Code Optimization, and Code Generation — was represented and enacted by students. They demonstrated how source code is processed step-by-step, mimicking the compiler's pipeline. This helped in visualizing the internal working of a compiler in a more tangible and fun way.

The role play was conducted successfully in **Room No. 410**, and the students showed excellent teamwork, coordination, and creativity in portraying each phase of the compiler.

Number of Participants: 18

List of Participating Students:

1. Faisal Khan
2. Ankit Pandey
3. Aditya Pratap Singh
4. Ankit Shukla
5. Avinash Gautam
6. Ankit Kumar
7. Aman Raza
8. Diwakar
9. Balram Kumar
10. Gunjan Kumar
11. Chandan Acharya
12. Aachal Sharma
13. Abhishek Kumar Soni
14. Ayush Bhatt
15. Guddu Kumar Manjhi
16. Ayush Singh
17. Aman Kumar
18. Abhishek Kumar

Attendance Sheet

AMBALIKA INSTITUTE OF MANAGEMENT & TECHNOLOGY, LUCKNOW

Role Play

Attendance Sheet

Course Name:	Compiler Design	Date:	22/8/2025
Branch:	CSE	Year:	
Section:	A	Instructor Name:	VIPIN RAWAT
Topic Name:	Phases of compiler		

Sr. No	Roll No.	Name of Student	Sign
1	2203630100065	Faisal Khan	Faisal.
2	2203630100030	Ankit Pandey	Ankit
3	2203630100015	Aditya pratap singh	Aditya
4	2203630100031	Mridul Shukla	Mridul
5	2203630100045	Avinash Gautam	Avinash
6	2203630100025	Ankit Kumar	Ankit
7	2203630100070	Aman Rana	Aman Rana
8	2203600100061	Diwakar Nath Ojha	Diwakar
9	2203630100053	Balram Kumar	Balram
10	2203630100070	Gunjan Kumar	Gunjan
11	2203630100055	Gaurav Adarsh	Gaurav
12	2203630100001	Anchal Sharma	Anchal
13	2203630100007	Abhishek Kr soni	Abhishek
14	2203630100046	Ayush Dehariya	Ayush
15	2203630100068	Guddu Km. Manjhi	Guddu
16	2203630100049	Ayush singh	Ayush
17	2203630100019	Aman kumar	Aman Kr.
18	2203630100005	Abhishek kumar	Abhishek

22/8/2025

ROLE –PLAY ACTIVITY

TEAM-1 AYUSH KUMAR 2202630100048
DIWAKAR KUMAR OJHA 2202630100064

TASK-LEXICAL ANALYSIS (PROBLEM GENERATION)

Problem - Tokenizing a simple Expression

you are designing a lexical Analyzer for a basic arithemalic language that supports the following :

- Integers (e.g., 42, 123)
- Arithemalic operators : +, -, *, /
- parenthesis : (,)
- whitespace (spaces or tabs, to be ignored)

Given the input string:

45 + 3 * (12 - 7)

Task :

manually Analyze the input and list all tokens produced by the lexical Analyzer . for each token, provide :

1. The lexme (the exact string matched)
2. The token type (e.g., Integer, operator, left_paren, Right_paren):

Balram kumar
Roll no.: 22036301 00053

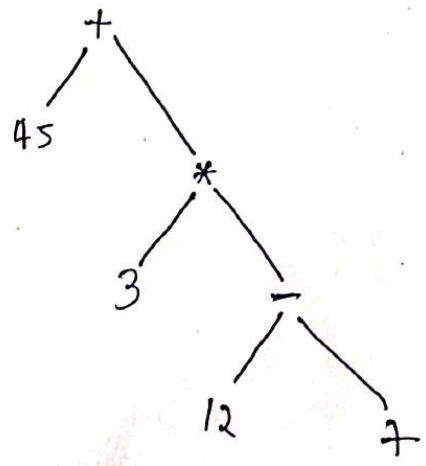
ROLE –PLAY ACTIVITY

TEAM-2

TASK- SYNTAX ANALYSIS OR PARSING

∴ Parse tree is constructed using tokens
↳ also check the Expression follow the grammar or not

(1) The expression starts with an integer literal (45).
(2) The + operator indicates an addition operation
(3) The next token is an integer literal (3) which is the operand for the multiplication operation
(4) The * operator indicates a multiplication operation
(5) The left parenthesis (opens a new expression)
(6) The integer literal (12) is the operand for the subtraction operation
(7) The - operator indicates a subtraction operation
(8) The integer literal (7) is the second operand for the subtraction operation
(9) The right parenthesis) closes the expression



Aman Raza
2203630100020

ROLE –PLAY ACTIVITY

TEAM-3

TASK- SEMANTIC ANALYSIS

1. Type Checking:
   - All operands are integers, which is valid for arithmetic operations.

2. Operator Application:
   - The operator "+", "*" and "-" are applied correctly between integers.

3. Parentheses Matching:
   - The parentheses "(" and ")" properly enclosed the subexpression 12-7, ensuring correct order of operations.

4. Operator Precedence:
   - The operator "*" has higher precedence, so 3 * (12-7) is computed before adding 45.

5. No Unused Tokens:
   - Every token is part of a valid expression and there are no missing or extra tokens.

Result:- The Expression is syntactically and sementically correct.

Faisal Ichan
2203630100065

ROLE –PLAY ACTIVITY

TEAM-4

TASK- INTERMEDIATE CODE GENERATION

Goal:- Break down the high-level expression into simpler instructions using temporary variables (t_1, t_2, t_3, etc). Each instruction typically has at most three addresses.

Steps

1. Subtraction (12-7)

   $t_1 = 12 - 7$

   - Store the result of 12-7 in t_1

2. Multiplication $3 * t_1$

   $t_2 = 3 * t_1$

   - multiply 3 by the result in t_1 and store it in t_2

3. Addition $45 + t_2$

   $t_3 = 45 + t_2$

   - Add 45 to the result in t_2 and store it in t_3

Result

- The final value of the original expression is now in t_3.

Aditya pratap singh
2203630100015

ROLE –PLAY ACTIVITY

TEAM-5

TASK- CODE OPTIMIZATION

- Optimized Code for Tokenization (python)

```
import re
Token_SPEC = [
    (r'\d+', 'INTEGER')
    (r'[+\-*/]', 'OPERATION),
    (r'[()]', 'PAREN'),
    (r'\s+', None)
]
def tokenize (expression):
    tokens = []
    For pattern, token_type in TOKEN_SPEC;
    For match in re.finditer(pattern, expression):
    if token_type;
    tokens.append((match.group(), token_type)
    return tokens
expression = "45 + 3* (12-7)"
tokens = tokenize (expression)
For lexeme, token_type in tokens:
    print (F" Lexeme : (Lexeme), Token Type
    (token_type)")
```

ANKIT SHUKLA
2203630100031

ROLE –PLAY ACTIVITY

TEAM-6

TASK- TARGET CODE GENERATION

Lexical Analyzer code in Python.

```
import re
Token Type = {
  'Integer': r'\d+'
  'Operator': r'[+\-*/]',
  'Left-Paren': r'(',
  'Right-Paren': r')',
  'Whitespace': r'\s+'
    Token-Type.items()
def tokenize(expression):
  tokens = []
re.finditer(Token-Regex, expression):
token-type = match.lastgroup
lexeme = match.group()
if token-type != 'Whitespace':
tokens.append((lexeme, token-type))
  return tokens.
```

Glimpses of Roleplay Activity

EVALUATION OF THE ACTIVITY

AMBALIKA INSTITUTE OF MANAGEMENT & TECHNOLOGY, LUCKNOW

Role Play

Evaluation Sheet

Course Name: Compiler Design | Date: 22/3/2025

Branch: CSE | Year: 3

Section: A | Instructor Name: VIPIN RAWAT

Topic Name: Phases of Compiler

Sr. No	Roll No.	Group	Name of Student	Role	Content Accuracy (5)	Presentation Skills (5)	Creativity (5)	Team Collaboration (5)	Total Score	Remarks
1	2203630100065	1	Faisal Khan	LEXICAL Analysis	4	5	4	4	17	
2	2203630100030	1	Ankit Pandey							
3	2203630100015	1	Aditya Pratap Singh							
4	2203630100031	2	Ankit Shukla	Syntax Analysis or Parsing	3	4	3	5	15	
5	2203630100045	2	Avinash gautam							
6	2203630100025	2	Ankit Kumar							
7	2203630100020	3	Aman Raza	semantic Analysis	3	3	4	4	14	
8	2203630100064	3	Diwakar Kr. Ojha							
9	2203630100053	3	Balram Kumar							
10	2203630100070	4	Gunjan Kumar	Intermediate code generation	4	4	3	4	15	
11	2203630100055	4	Chandan acharya							
12	2203630100001	4	AAnchal Sharma							
13	2203630100007	5	Abishek Km. Soni	code optimization	4	4	4	4	16	
14	2203630100046	5	Ayush bhutt							
15	2203630100068	5	Guddu Kr. manjhi							
16	2203630100019	6	Ayush singh	Target code generation	4	3	4	5	16	
17	2203630100019	6	Aman kumar							
18	2203630100005	6	Abhishek kumar							

22.3.25

22/3/25

Outcome of the Activity:

- Students gained a deeper understanding of compiler phases through experiential learning.
- Communication and presentation skills were enhanced.
- Learners became more engaged and motivated to explore the subject further.
- The activity promoted collaborative learning and peer-to-peer teaching.

Conclusion:

The role play on *"Phases of a Compiler"* proved to be a highly effective pedagogical tool. It not only enriched the learning experience but also fostered greater interest in Compiler Design. Such activities are highly recommended as regular part of curriculum delivery to transform passive learning into active exploration.