PRACTICAL FILE COMPILER DESIGN

AMITY UNIVERSITY
PUNJAB

B. TECH-CSE-B

SEMESTER-6th

SUBMITTED TO:

DR. LAW KUMAR SINGH
ASET, AMITY UNIVERSITY
MOHALI

SUBMITTED BY:

KAASHIF MATTO
B.Tech CSE – 6TH SEM
A25305222145

INDEX

Practical 1: Calculator program in C

Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    char operator;
    double num1, num2, result;
    printf("Enter operator (+, -, *, /): ");
    scanf(" %c", &operator);
    printf("Enter two numbers: ");
    scanf("%lf %lf", &num1, &num2);
    switch (operator) {
        case '+':
            result = num1 + num2;
            printf("%.2lf + %.2lf = %.2lf\n", num1, num2, result);
            break;
        case '-':
            result = num1 - num2;
            printf("%.2lf - %.2lf = %.2lf\n", num1, num2, result);
            break;
        case '*':
            result = num1 * num2;
            printf("%.2lf * %.2lf = %.2lf\n", num1, num2, result);
            break;
        case '/':
            if (num2 != 0) {
                result = num1 / num2;
                printf("%.2lf / %.2lf = %.2lf\n", num1, num2, result);
            } else {
                printf("Error: Division by zero!\n");
                return 1;
            }
            break;
        default:
            printf("Error: Invalid operator!\n");
            return 1;
    }
    return 0;
}
```

Output:

```
Enter operator (+, -, *, /): *
Enter two numbers: 80 80
80.00 * 80.00 = 6400.00
```

Practical 2: Matrix addition code in C

Code:

```
#include <stdio.h>
#define ROWS 3
#define COLS 3
int main() {
    int matrix1[ROWS][COLS] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int matrix2[ROWS][COLS] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};
    int sum[ROWS][COLS];
    printf("Matrix 1:\n");
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", matrix1[i][j]);
        }
        printf("\n");
    }
    printf("Matrix 2:\n");
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", matrix2[i][j]);
        }
        printf("\n");
    }
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
    printf("Sum of the matrices:\n");
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", sum[i][j]);
        }
        printf("\n");
    }
```

```
    return 0;
}
```

Output:

```
Matrix 1:
1 2 3
4 5 6
7 8 9
Matrix 2:
9 8 7
6 5 4
3 2 1
Sum of the matrices:
10 10 10
10 10 10
10 10 10
```

Practical 3: Matrix multiplication code in C

Code:

```
#include <stdio.h>
#define ROWS1 3
#define COLS1 3
#define ROWS2 3
#define COLS2 3
int main() {
    int matrix1[ROWS1][COLS1] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int matrix2[ROWS2][COLS2] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};
    int product[ROWS1][COLS2];
    if (COLS1 != ROWS2) {
        printf("Error: Matrices cannot be multiplied!\n");
        return 1;
    }
    for (int i = 0; i < ROWS1; i++) {
        for (int j = 0; j < COLS2; j++) {
            product[i][j] = 0;
        }
    }
```

```
    for (int i = 0; i < ROWS1; i++) {
        for (int j = 0; j < COLS2; j++) {
            for (int k = 0; k < COLS1; k++) {
                product[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
    printf("Matrix 1:\n");
    for (int i = 0; i < ROWS1; i++) {
        for (int j = 0; j < COLS1; j++) {
            printf("%d ", matrix1[i][j]);
        }
        printf("\n");
    }
    printf("Matrix 2:\n");
    for (int i = 0; i < ROWS2; i++) {
        for (int j = 0; j < COLS2; j++) {
            printf("%d ", matrix2[i][j]);
        }
        printf("\n");
    }
    printf("Product of the matrices:\n");
    for (int i = 0; i < ROWS1; i++) {
        for (int j = 0; j < COLS2; j++) {
            printf("%d ", product[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
Matrix 1:
1 2 3
4 5 6
7 8 9
Matrix 2:
9 8 7
6 5 4
3 2 1
Product of the matrices:
30 24 18
84 69 54
138 114 90
```

Practical 4: WAP to Check String is Constant or Not

Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int isConstant(char *str) {
    int i = 0;
    if (str[0] == '-' || str[0] == '+') {
        i = 1;
    }
    for (; str[i] != '\0'; i++) {
        if (!isdigit(str[i])) {
            return 0;
        }
    }
    return 1;
}
int main() {
    char input[100];
    printf("Enter a string: ");
    scanf("%s", input);
    if (isConstant(input)) {
        printf("\"%s\" is a constant.\n", input);
    } else {
        printf("\"%s\" is not a constant.\n", input);
```

```
    }
    return 0;
}
```

Output:

```
Enter a string: 1325
"1325" is a constant.
```

Practical 5: Count No. of Lines and Spaces in C

Code:

```
#include <stdio.h>
#include <string.h>
int main() {
    char ch;
    int lines = 0, spaces = 0;
    char line[256];
    printf("Enter text (Enter # on a new line to end):\n");
    while (fgets(line, sizeof(line), stdin) != NULL) {
        if (strcmp(line, "#\n") == 0) {
            break;
        }
        for (int i = 0; line[i] != '\0'; i++) {
            ch = line[i];
            if (ch == '\n') {
                lines++;
            } else if (ch == ' ') {
                spaces++;
            }
        }
    }
    printf("Number of lines: %d\n", lines);
    printf("Number of spaces: %d\n", spaces);
    return 0;
}
```

Output:

```
Enter text (Enter # on a new line to end):
My name is kaashif
I am Studying B.Tech CSE
I love to play football
#
Number of lines: 3
Number of spaces: 14
```

Practical 6: WAP to check identifiers in C program

Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int isValidIdentifier(char *str)
{
    if (!isalpha(str[0]) && str[0] != '_')
    {
        return 0;
    }
    for (int i = 1; str[i] != '\0'; i++)
    {
        if (!isalnum(str[i]) && str[i] != '_')
        {
            return 0;
        }
    }
    return 1;
}
int main()
{
    char input[100];
    printf("Enter a string: ");
    scanf("%s", input);
    if (isValidIdentifier(input)) {
        printf("\"%s\" is a valid identifier.\n", input);
    } else {
        printf("\"%s\" is not a valid identifier.\n", input);
    }
    return 0;
```

}

Output:

```
Enter a string: sbc1323
"sbc1323" is a valid identifier.
```

Practical 7: WAP to check keywords in program

Code:

```
#include <stdio.h>
#include <string.h>
const char *keywords[] = {
    "auto", "break", "case", "char", "const", "continue", "default", "do", "double",
"else",
    "enum", "extern", "float", "for", "goto", "if", "int", "long", "register",
"return",
    "short", "signed", "sizeof", "static", "struct", "switch", "typedef", "union",
"unsigned", "void",
    "volatile", "while"
};
#define TOTAL_KEYWORDS (sizeof(keywords) / sizeof(keywords[0]))

int isKeyword(const char *word) {
    for (int i = 0; i < TOTAL_KEYWORDS; i++) {
        if (strcmp(word, keywords[i]) == 0) {
            return 1;
        }
    }
    return 0;
}
int main() {
    char word[50];
    printf("Enter a word to check if it's a C keyword: ");
    scanf("%s", word);
    if (isKeyword(word)) {
        printf("%s is a C keyword.\n", word);
    } else {
        printf("%s is not a C keyword.\n", word);
    }
    return 0;
}
```

Output:

```
Enter a word to check if it's a C keyword: int
int is a C keyword.
```

Practical 8: Write a menu based program to check identifiers, space and constant in C program (Lexical Analyzer)

Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
char *keywords[] = {
    "int", "float", "return", "if", "else", "while", "for", "do",
    "switch", "case", "break", "continue", "char", "double", "long",
    "short", "struct", "typedef", "union", "unsigned", "void", "static",
    "default", "const", "sizeof", "volatile", "enum", "goto"
};
int num_keywords = sizeof(keywords) / sizeof(keywords[0]);
int isKeyword(char *word) {
    for (int i = 0; i < num_keywords; i++) {
        if (strcmp(word, keywords[i]) == 0)
            return 1;
    }
    return 0;
}
int isIdentifier(char *word) {
    if (!isalpha(word[0]) && word[0] != '_')
        return 0;
    for (int i = 1; word[i] != '\0'; i++) {
        if (!isalnum(word[i]) && word[i] != '_')
            return 0;
    }
    return 1;
}
int isNumber(char *word) {
    for (int i = 0; word[i] != '\0'; i++) {
        if (!isdigit(word[i]))
            return 0;
    }
    return 1;
}
```

```
int isOperator(char ch) {
    char operators[] = "+-*/%=<>!&|^";
    for (int i = 0; i < strlen(operators); i++) {
        if (ch == operators[i])
            return 1;
    }
    return 0;
}
void lexicalAnalyzer(char *input) {
    char word[50];
    int index = 0;
    for (int i = 0; input[i] != '\0'; i++) {
        char ch = input[i];
        if (isalnum(ch) || ch == '_') {
            word[index++] = ch;
        } else {
            if (index > 0) {
                word[index] = '\0';
                index = 0;
                if (isKeyword(word))
                    printf("Keyword: %s\n", word);
                else if (isNumber(word))
                    printf("Number: %s\n", word);
                else if (isIdentifier(word))
                    printf("Identifier: %s\n", word);
                else
                    printf("Unknown Token: %s\n", word);
            }
            if (isOperator(ch)) {
                printf("Operator: %c\n", ch);
            }
        }
    }
}
int main() {
    int choice;
    char input[100];
    do {
        printf("\nLexical Analyzer Menu\n");
        printf("1. Enter a statement\n");
        printf("2. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        getchar();
        switch (choice) {
            case 1:
                printf("Enter the statement: ");
                fgets(input, sizeof(input), stdin);
                input[strcspn(input, "\n")] = 0; // Remove newline character
                lexicalAnalyzer(input);
                break;
            case 2:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 2);
    return 0;
}
```

Output:

```
Lexical Analyzer Menu
1. Enter a statement
2. Exit
Enter your choice: 1
Enter the statement: int main ( int a = 10 )
Keyword: int
Identifier: main
Keyword: int
Identifier: a
Operator: =
Number: 10
```

Practical 9: WAP for left recursion in C

Code:

```
#include <stdio.h>
#include <string.h>
#define MAX_RULES 10
#define MAX_SYMBOLS 10
```

```
int main() {
    char rules[MAX_RULES][MAX_SYMBOLS];
    int numRules;
    printf("Enter the number of grammar rules: ");
    scanf("%d", &numRules);
    printf("Enter the grammar rules (e.g., A->Ab|c):\n");
    for (int i = 0; i < numRules; i++) {
        scanf("%s", rules[i]);
    }
    printf("Removing left recursion:\n");
    for (int i = 0; i < numRules; i++) {
        char nonTerminal = rules[i][0];
        int j = 3;
        if (rules[i][j] == nonTerminal) {
            printf("Left recursion found in rule: %s\n", rules[i]);
            char a_part[MAX_SYMBOLS] = "";
            char b_part[MAX_SYMBOLS] = "";
            int k = j + 1;
            int a_index = 0;
            int b_index = 0;
            int is_a = 0;
            while (rules[i][k] != '\0') {
                if (rules[i][k] == '|') {
                    is_a = 0;
                    k++;
                    continue;
                }
                if (is_a == 0 && rules[i][k] != nonTerminal) {
                    b_part[b_index++] = rules[i][k];
                } else if (is_a == 1) {
                    a_part[a_index++] = rules[i][k];
                }
                if (rules[i][k] == nonTerminal) {
                    is_a = 1;
                }
                k++;
            }
            a_part[a_index] = '\0';
            b_part[b_index] = '\0';
            printf("  %c  -> %s%c'\n", nonTerminal, b_part, nonTerminal);
            printf("  %c' -> %s%c' | e\n", nonTerminal, a_part, nonTerminal);
        } else {
            printf("No left recursion found in rule: %s\n", rules[i]);
```

```
        }
    }
    return 0;
}
```

Output:

```
Enter the number of grammar rules: 2
Enter the grammar rules (e.g., A->Ab|c):
A->Ab|c
B->Bb|z
Removing left recursion:
Left recursion found in rule: A->Ab|c
  A  -> bcA'
  A' -> A' | e
Left recursion found in rule: B->Bb|z
  B  -> bzB'
  B' -> B' | e
```