CD LAB Study Material

Lab 1

symbol_table.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Node {
    char data[500];
    struct Node *next;
};

struct Node *start = NULL;

int main() {
    int choice;

    while (1) {
        printf("\nSymbol Table Management:\n");
        printf("1. Insert\n2. Display\n3. Search\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                struct Node *temp = malloc(sizeof(struct Node));
                printf("Enter identifier: ");
                scanf("%s", temp->data);
                temp->next = start;
                start = temp;
                printf("Inserted.\n");
                break;
            }
            case 2: {
                if (!start) {
                    printf("Symbol table is empty.\n");
                    break;
                }
                printf("\nIdentifier\tAddress\n------------------------\n");
                for (struct Node *cur = start; cur; cur = cur->next)
                    printf("%s\t\t%p\n", cur->data, (void *)cur);
                break;
            }
            case 3: {
                char key[500];
                printf("Enter identifier to search: ");
                scanf("%s", key);
                struct Node *cur = start;
```

```
                while (cur) {
                    if (strcmp(cur->data, key) == 0) {
                        printf("Found at address: %p\n", (void *)cur);
                        break;
                    }
                    cur = cur->next;
                }
                if (!cur) printf("Not found.\n");
                break;
            }
            case 4:
                while (start) {
                    struct Node *temp = start;
                    start = start->next;
                    free(temp);
                }
                printf("Memory freed. Exiting...\n");
                return 0;
            default:
                printf("Invalid choice.\n");
        }
    }
}
```

Lab 2

lexical_analyzer.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

char * operators = "+-*/=<>%";
char * delimiters = "(){},;[]";
char * keywords[] = {
  "int",
  "void",
  "main",
  "char",
  "float"
};

char identifiers[20][20], operatorsList[20][20], delimitersList[20],
keywordsList[20][20];
char literals[20][20], constants[20][20];

int idCount = 0, opCount = 0, delCount = 0, keyCount = 0, litCount = 0, constCount
= 0;
```

```c
int isOperator(char ch) {
  return strchr(operators, ch) != NULL;
}

int isDelimiter(char ch) {
  return strchr(delimiters, ch) != NULL;
}

int isKeyword(char * str) {
  for (int i = 0; i < 5; i++)
    if (strcmp(str, keywords[i]) == 0)
      return 1;
  return 0;
}

void analyzeLexical(char * str) {
  int i = 0, j;
  char buffer[50];

  while (str[i] != '\0') {
    if (isalpha(str[i])) { // Identifier or Keyword
      j = 0;
      while (isalpha(str[i]) || isdigit(str[i]) || str[i] == '_')
        buffer[j++] = str[i++];
      buffer[j] = '\0';

      if (isKeyword(buffer))
        strcpy(keywordsList[keyCount++], buffer);
      else
        strcpy(identifiers[idCount++], buffer);
    } else if (isdigit(str[i])) { // Constant
      j = 0;
      while (isdigit(str[i]))
        buffer[j++] = str[i++];
      buffer[j] = '\0';
      strcpy(constants[constCount++], buffer);
    } else if (str[i] == '"') { // Literal String
      j = 0;
      buffer[j++] = str[i++];
      while (str[i] != '"' && str[i] != '\0')
        buffer[j++] = str[i++];
      buffer[j++] = str[i++];
      buffer[j] = '\0';
      strcpy(literals[litCount++], buffer);
    } else if (isOperator(str[i])) { // Operator
      j = 0;
      while (isOperator(str[i]))
        buffer[j++] = str[i++];
      buffer[j] = '\0';
      strcpy(operatorsList[opCount++], buffer);
    } else if (isDelimiter(str[i])) { // Delimiter
      delimitersList[delCount++] = str[i++];
    } else {
      i++;
    }
```

```
    }
  }

  void printResults() {
    printf("\n--- Lexical Analysis Results ---\n");

    printf("\n Identifiers:\n");
    for (int i = 0; i < idCount; i++)
      printf("  %s\n", identifiers[i]);

    printf("\n Keywords:\n");
    for (int i = 0; i < keyCount; i++)
      printf("  %s\n", keywordsList[i]);

    printf("\n Operators:\n");
    for (int i = 0; i < opCount; i++)
      printf("  %s\n", operatorsList[i]);

    printf("\n Delimiters:\n");
    for (int i = 0; i < delCount; i++)
      printf("  %c\n", delimitersList[i]);

    printf("\n Constants:\n");
    for (int i = 0; i < constCount; i++)
      printf("  %s\n", constants[i]);

    printf("\n Literals:\n");
    for (int i = 0; i < litCount; i++)
      printf("  %s\n", literals[i]);

    printf("\n-------------------------------\n");
  }

  int main() {
    char str[1000];

    printf("Enter the code snippet:\n");
    fgets(str, sizeof(str), stdin);

    analyzeLexical(str);
    printResults();

    return 0;
  }
```

Lab 3

echo.l

```
%%
```

```
.    ECHO;  // Echo any character
\n   ECHO;  // Echo newline characters
%%

int yywrap(void) {
    return 1;
}

int main() {
    printf("Start typing:\n");
    yylex();
    return 0;
}
```

identifiers.l

```
%%

[a-zA-Z_][a-zA-Z0-9_]*  { printf("It is an Identifier\n"); }

.  { printf("It is not an Identifier\n"); }

%%

int yywrap(void) {
    return 1;
}

int main() {
    yylex();
    return 0;
}
```

line_number.l

```
%{
int l = 1;
%}

%%

^(.*)\n  { printf("%d\t%s", l++, yytext); }  // Print line number and text

%%

int yywrap(void) {
```

```
    return 1;
}

int main() {
    yyin = fopen("input.txt", "r");
    yylex();
    return 0;
}
```

vowel_consonant.l

```
%{
int vowels = 0, consonants = 0;
%}

%%

[aeiouAEIOU] { vowels++; printf("Vowel "); }
[a-zA-Z]     { consonants++; printf("Consonant "); }

[^a-zA-Z]    { printf("\n"); }

%%

int yywrap(void) {
    return 1;
}

int main() {
    printf("Enter text (Ctrl+D to stop input on Linux/macOS, Ctrl+Z on
Windows):\n");
    yylex();
    printf("\nTotal Vowels: %d\nTotal Consonants: %d\n", vowels, consonants);
    return 0;
}
```

input.txt

```
Hello World!
This is a test file.
Lexical Analysis in action.
```

Lab 4

lexical_analyzer.l

```
%{
int k = 0, i = 0, e = 0, c = 0, d = 0;
%}

%%

int|float|double|void|scanf|printf|return { printf("Keyword "); i++; }
[a-zA-Z_][a-zA-Z0-9_]* { printf("Identifier "); k++; }
[0-9]+ { printf("Digit "); e++; }
[+\-*/%=] { printf("Operator "); c++; }
[(){},;[\]] { printf("Delimiter "); d++; }

%%

int yywrap(void) {
    return 1;
}

int main() {
    yyin = fopen("lexy.txt", "r");
    yylex();

    printf("\n\nIdentifier count: %d\n", k);
    printf("Keyword count: %d\n", i);
    printf("Digit count: %d\n", e);
    printf("Operator count: %d\n", c);
    printf("Delimiter count: %d\n", d);

    fclose(yyin);
    return 0;
}
```

lexy.txt

```
int a = 5;
int b = 5;
int c = a + b;
```

Lab 5

valid.l

```
%{
#include "y.tab.h"
#include <stdlib.h>
#include <stdio.h>

void yyerror(const char *);
%}

%%

[a-z] {
    printf("Identifier: %s\n", yytext);
    return VARIABLE;
}

[0-9]+ {
    yylval = atoi(yytext);
    return INTEGER;
}

[-+()=*/\n] { return *yytext; }

[ \t] ; /* Ignore whitespace */

. {
    yyerror("Unknown character found!");
}

%%

int yywrap(void) {
    return 1;
}
```

valid.y

```
%{
#include <stdio.h>
#include <stdlib.h>

void yyerror(const char *s);
int yylex(void);
int sym[26];  // Symbol table
%}

%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
```

```
%%
program:
    program statement '\n'
    | /* NULL */
    ;

statement:
    expression { printf(" Valid Expression\n"); }
    | VARIABLE '=' expression { printf(" Valid Assignment\n"); sym[$1] = $3; }
    ;

expression:
    INTEGER { $$ = $1; }
    | VARIABLE { $$ = sym[$1]; }
    | expression '+' expression { $$ = $1 + $3; }
    | expression '-' expression { $$ = $1 - $3; }
    | expression '*' expression { $$ = $1 * $3; }
    | expression '/' expression {
        if ($3 == 0) {
            yyerror(" Error: Division by zero!");
        } else {
            $$ = $1 / $3;
        }
      }
    | '(' expression ')' { $$ = $2; }
    ;

%%

void yyerror(const char *s) {
    fprintf(stderr, " Invalid Expression: %s\n", s);
    exit(1);
}

int main(void) {
    printf("Enter an expression to validate:\n");
    return yyparse();
}
```

Lab 6

calc.l

```
%{
    #include "y.tab.h"
    #include <stdlib.h>

    void yyerror(char *);
%}
```

```
%%

[a-z] {
    printf("Identifier: %s\n", yytext);
    return VARIABLE;
}

[0-9]+ {
    yylval = atoi(yytext);
    return INTEGER;
}

[-+()=*/\n] { return *yytext; }

[ \t] ; /* Ignore whitespace */

. { yyerror("Unknown character"); }

%%

int yywrap(void) {
    return 1;
}
```

calc.y

```
%{
    #include <stdio.h>
    #include <stdlib.h>

    void yyerror(char *);
    int yylex(void);
    int sym[26]; // Array to store variable values
%}

%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'

%%
program:
    program statment '\n'
    | /* NULL */
    ;

statment:
    expression { printf("Result = %d\n", $1); }
    | VARIABLE '=' expression { sym[$1] = $3; printf("Assigned %d to variable\n",
$3); }
```

```
    ;

expression:
    INTEGER { $$ = $1; }
    | VARIABLE { $$ = sym[$1]; }
    | expression '+' expression { $$ = $1 + $3; }
    | expression '-' expression { $$ = $1 - $3; }
    | expression '*' expression { $$ = $1 * $3; }
    | expression '/' expression { $$ = $1 / $3; }
    | '(' expression ')' { $$ = $2; }
    ;
%%

void yyerror(char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main(void) {
    return yyparse();
}
```

Lab 7

validator.l

```
%{
#include "y.tab.h"
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

void yyerror(const char *);
%}

%%

[a-zA-Z][a-zA-Z0-9]* {
    printf(" Valid Identifier: %s\n", yytext);
    return VARIABLE;
}

[0-9]+ {
    printf("Valid Number: %s\n", yytext);
    return INTEGER;
}

[-+()=*/] { return *yytext; }

[ \t] ; /* Ignore whitespace */
```

```
[0-9]+[a-zA-Z]+ {
    yyerror("Invalid Identifier: Cannot mix numbers and letters incorrectly!");
}

. {
    yyerror(" Syntax Error: Unknown character!");
}

%%

int yywrap(void) {
    return 1;
}
```

validator.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

void yyerror(const char *);
int yylex(void);
%}

%token VARIABLE INTEGER
%left '+' '-'
%left '*' '/'

%%
program:
    program statement '\n'
    | /* NULL */
    ;

statement:
    expression { printf("Valid Expression\n"); }
    ;

expression:
    VARIABLE
    | INTEGER
    | expression '+' expression
    | expression '-' expression
    | expression '*' expression
    | expression '/' expression
    | '(' expression ')'
    ;

%%
```

```
void yyerror(const char *s) {
    fprintf(stderr, "Syntax Error: %s\n", s);
    exit(1);
}

int main(void) {
    printf("Enter an expression to validate:\n");
    return yyparse();
}
```

Lab 8

type_checking.c

```
#include <stdio.h>
#include <string.h>

#define MAX_VARS 50

typedef struct {
    int type;  // 1 = int, 2 = float
    char var[10];
} SymTable;

SymTable sT[MAX_VARS];
int count = 0;

void addVariables(char *line) {
    char type[10], var[10];
    sscanf(line, "%s", type);
    line += strlen(type) + 1;

    char *token = strtok(line, ", ;");
    while (token) {
        sT[count].type = (strcmp(type, "int") == 0) ? 1 : 2;
        strcpy(sT[count++].var, token);
        token = strtok(NULL, ", ;");
    }
}

int getType(char *var) {
    for (int i = 0; i < count; i++) {
        if (strcmp(sT[i].var, var) == 0)
            return sT[i].type;
    }
    return -1; // Undeclared variable
}

int checkExpression(char *expr) {
```

```
    char var[10];
    int expectedType = 0;

    char *token = strtok(expr, " +=;");
    while (token) {
        int type = getType(token);
        if (type == -1) continue; // Ignore non-variable tokens

        if (expectedType == 0)
            expectedType = type;
        else if (expectedType != type)
            return 0; // Type mismatch

        token = strtok(NULL, " +=;");
    }
    return 1;
}

int main() {
    int N;
    char line[50], expr[50];

    printf("Enter number of declarations: ");
    scanf("%d", &N);
    getchar(); // Consume newline

    while (N--) {
        fgets(line, sizeof(line), stdin);
        addVariables(line);
    }

    printf("Enter expression: ");
    fgets(expr, sizeof(expr), stdin);

    printf(checkExpression(expr) ? "Correct\n" : "Semantic error\n");

    return 0;
}
```

Lab 9

frontend.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

#define MAX_QUADS 100
```

```c
// Structure for Three-Address Code (TAC)
typedef struct {
    int pos;
    char op[5];
    char arg1[10];
    char arg2[10];
    char result[10];
} TAC;

TAC tac[MAX_QUADS];
int tempVarCount = 0, tacIndex = 0, labelCount = 0;

// Generate new temporary variable
void newTempVar(char *temp) {
    sprintf(temp, "t%d", tempVarCount++);
}

// Generate new label
void newLabel(char *label) {
    sprintf(label, "L%d", labelCount++);
}

// Generate TAC instruction
void generateTAC(char *op, char *arg1, char *arg2, char *result) {
    tac[tacIndex].pos = tacIndex + 1;
    strcpy(tac[tacIndex].op, op);
    strcpy(tac[tacIndex].arg1, arg1);
    strcpy(tac[tacIndex].arg2, arg2);
    strcpy(tac[tacIndex].result, result);
    tacIndex++;
}

// Process an assignment statement (e.g., a = 5 + b * c)
void processAssignment(char *line) {
    char left[10], operand1[10], operand2[10], op[3], temp[10];
    int i = 0, j = 0;

    // Extract left-hand side
    while (line[i] != '=' && line[i] != '\0') {
        left[j++] = line[i++];
    }
    left[j] = '\0';
    i++; // Skip '='

    // Process right-hand side
    j = 0;
    while (isalnum(line[i])) operand1[j++] = line[i++];
    operand1[j] = '\0';

    if (line[i] == '\0') {
        generateTAC(":=", operand1, "", left);
        return;
    }

    op[0] = line[i++];
```

```
    op[1] = '\0';

    j = 0;
    while (isalnum(line[i])) operand2[j++] = line[i++];
    operand2[j] = '\0';

    newTempVar(temp);
    generateTAC(op, operand1, operand2, temp);
    generateTAC(":=", temp, "", left);
}

// Process an if-condition (e.g., if (a > 10))
void processIfCondition(char *line) {
    char operand1[10], operand2[10], op[3], temp[10], label[10];
    int i = 3, j = 0; // Skip "if ("

    // Extract first operand
    while (isalnum(line[i])) operand1[j++] = line[i++];
    operand1[j] = '\0';

    // Extract operator
    op[0] = line[i++];
    if (line[i] == '=' || line[i] == '<' || line[i] == '>') {
        op[1] = line[i++];
        op[2] = '\0';
    } else {
        op[1] = '\0';
    }

    // Extract second operand
    j = 0;
    while (isalnum(line[i])) operand2[j++] = line[i++];
    operand2[j] = '\0';

    // Generate TAC for condition
    newTempVar(temp);
    generateTAC(op, operand1, operand2, temp);

    // Generate TAC for jump
    newLabel(label);
    generateTAC("if", temp, "goto", label);
}

// Print generated TAC
void printTAC() {
    printf("\n%-5s %-5s %-10s %-10s %-10s\n", "Pos", "Oper", "Arg1", "Arg2",
"Result");
    printf("-------------------------------------------------\n");

    for (int i = 0; i < tacIndex; i++) {
        printf("%-5d %-5s %-10s %-10s %-10s\n",
               tac[i].pos, tac[i].op, tac[i].arg1, tac[i].arg2, tac[i].result);
    }
}
```

```
// Main function to parse a C-style code
int main() {
    char code[10][50];
    FILE *file = fopen("input.txt", "r");
    char line[50];
    int lineCount = 0;

    while (fgets(line, sizeof(line), file)) {
        line[strcspn(line, "\n")] = '\0';
        strcpy(code[lineCount], line);
        lineCount++;
    }
    fclose(file);
    // Process each line
    for (int i = 0; i < lineCount; i++) {
        printf("%s\n", code[i]);
        if (strstr(code[i], "if") != NULL)
            processIfCondition(code[i]);
        else if (strstr(code[i], "=") != NULL)
            processAssignment(code[i]);
    }

    printTAC();

    return 0;
}
```

input.txt

```
a=5
b=a+3
c=b*2
if(c>10)
```

Lab 10

code_optimization.c

```
#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>
#include <string.h>

#define MAX_LINE 256

int main()
```

```
{
    char declared[26] = {0}; // Track declared variables (a-z)
    char used[26] = {0};     // Track used variables
    char line[MAX_LINE];
    FILE *file;

    file = fopen("test.txt", "r");
    if (!file) {
        printf("Error opening file\n");
        return 1;
    }

    printf("Program:\n");

    // Process each line
    while (fgets(line, MAX_LINE, file)) {
        printf("%s", line);

        // Create a clean version without whitespace
        size_t len = strlen(line);
        char cleaned[MAX_LINE] = {0};
        int cleanIndex = 0;

        for (int i = 0; i < len; i++) {
            if (!isspace(line[i])) {
                cleaned[cleanIndex++] = line[i];
            }
        }

        // Check for declarations (format: "a;")
        if (strlen(cleaned) == 2 && isalpha(cleaned[0]) && cleaned[1] == ';') {
            char var = tolower(cleaned[0]);
            if (var >= 'a' && var <= 'z') {
                declared[var - 'a'] = 1;
            }
        }

        // Check for assignments (format: "a=34;")
        for (int i = 0; i < strlen(cleaned) - 1; i++) {
            if (cleaned[i + 1] == '=' && isalpha(cleaned[i])) {
                char var = tolower(cleaned[i]);
                if (var >= 'a' && var <= 'z') {
                    used[var - 'a'] = 1;
                }
            }
        }
    }

    // Print unused variables
    printf("\nThe unused variables are: ");
    bool foundUnused = false;

    for (int i = 0; i < 26; i++) {
        if (declared[i] && !used[i]) {
            printf("%c ", 'a' + i);
```

```
            foundUnused = true;
        }
    }

    if (!foundUnused) {
        printf("None");
    }
    printf("\n");

    fclose(file);
    return 0;
}
```

test.txt

```
a;
b;
c;
a;
a = 34;
c = 78;
j = 45;
r = 30;
```

Lab 11

stack_allocation.c

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 5

int stack[SIZE], top = -1;

void push(int item) {
    if (top == SIZE - 1) {
        printf("\nStack is Full! Overflow!\n");
        return;
    }
    stack[++top] = item;
    printf("\nPushed: %d\n", item);
}

int pop() {
    if (top == -1) {
        printf("\nStack is Empty! Underflow!\n");
```

```
        return -1;
    }
    return stack[top--];
}

void display() {
    if (top == -1) {
        printf("\nStack is Empty!\n");
        return;
    }
    printf("\nStack contents:\n");
    for (int i = top; i >= 0; i--)
        printf("%d\n", stack[i]);
}

int main() {
    int choice, item;

    printf("\nStack Implementation using Array\n");

    while (1) {
        printf("\nMenu:\n1. Push\n2. Pop\n3. Display\n4. Exit\nEnter your choice:
");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("\nEnter item to push: ");
                scanf("%d", &item);
                push(item);
                break;
            case 2:
                item = pop();
                if (item != -1)
                    printf("\nPopped: %d\n", item);
                break;
            case 3:
                display();
                break;
            case 4:
                printf("\nExiting program...\n");
                exit(0);
            default:
                printf("\nInvalid choice! Try again.\n");
        }
    }

    return 0;
}
```

Lab 12

backend_compiler.c

```c
#include <stdio.h>
#include <string.h>

int main() {
    int n, reg = 1;
    char inp[100][100];

    printf("Enter the no of statements: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
        scanf("%s", inp[i]);

    for (int i = 0; i < n; i++) {
        char dest = inp[i][0];  // Variable to store result
        char op1 = inp[i][2];   // First operand
        char op = inp[i][3];    // Operator (+, -, *, /)
        char op2 = inp[i][4];   // Second operand

        printf("LOAD R%d %c\n", reg, op1);
        int r1 = reg++;

        printf("LOAD R%d %c\n", reg, op2);
        int r2 = reg++;

        if (op == '+')
            printf("ADD R%d R%d\n", r1, r2);
        else if (op == '-')
            printf("SUB R%d R%d\n", r1, r2);
        else if (op == '*')
            printf("MUL R%d R%d\n", r1, r2);
        else if (op == '/')
            printf("DIV R%d R%d\n", r1, r2);

        printf("STORE %c R%d\n", dest, r1);
    }

    return 0;
}
```