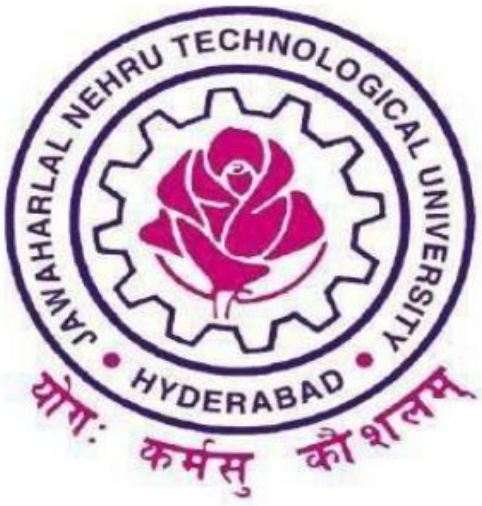
Jawaharlal Nehru Technological University Hyderabad

University College of Engineering Science and Technology Hyderabad

Department of Computer Science and Engineering

CERTIFICATE



This is to certify that _____ of B. TECH IV year I Semester bearing Hall-ticket number _____ has fulfilled his **COMPILER DESIGN LAB** record for the academic year 2025-2026.

Signature of the Head of the Department *Signature of the staff member*

*Date of Examination*_____

Internal Examiner ***External Examiner***

INDEX

1. **Develop a lexical analyzer to recognize a few patterns inc (ex. Identifiers, constants, comments, operators etc.)**

CODE:

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

char keywords[32][10] = {
   "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"
};

int isKeyword(char *str){
   for(int i=0;i<32;i++){
      if(strcmp(keywords[i],str)==0){
         return 1;
      }
   }
   return 0;
}

int isOperator(char ch){
   if(ch=='+'||ch=='-
'||ch=='*'||ch=='/'||ch=='='||ch=='<'||ch=='>'||ch=='&'||ch=='|'||ch=='!'||ch=='
%'){
      return 1;
   }
   return 0;
}

int isDelimiter(char ch){

if(ch==','||ch=='('||ch==')'||ch=='{'||ch=='}'||ch=='['||ch==']'||ch==';'||ch=='\n'|
|ch=='\t'||ch==' '){
      return 1;
   }
   return 0;
```

```
}

void analyze(char *input){
   int i=0;
   while(input[i]!='\0'){
      char current_char = input[i];

      if(isOperator(current_char)){
         printf("[OPERATOR]: %c\n",current_char);
         i++;
         continue;
      }

      if(isDelimiter(current_char)){
         printf("[DELIMITER]: %c\n",current_char);
         i++;
         continue;
      }

      if(isalpha(current_char)||current_char=='_'){
         char buffer[50];
         int j=0;
         while(isalnum(input[i])||input[i]=='_'){
            buffer[j++]=input[i++];
         }
         buffer[j]='\0';
         if(isKeyword(buffer)){
            printf("[KEYWORD]: %s\n",buffer);
         }
         else{
            printf("[IDENTIFIER]: %s\n",buffer);
         }
         continue;
      }

      if(isdigit(current_char)){
         char buffer[50];
         int j=0;
         while(isdigit(input[i])){
            buffer[j++]=input[i++];
```

```
        }
        buffer[j]='\0';
        printf("[CONSTANT (Integer)]: %s\n",buffer);
      }

      if(current_char=='/'&&input[i+1]=='/'){
        printf("{COMMENT}: ");
        i+=2;
        while(input[i]!='\n'&&input[i]!='\0'){
          printf("%c",input[i]);
          i++;
        }
        printf("\n");
        continue;
      }
      i++;
    }
}

int main(){
    char code[]="int x=10; //start\nif (x>5) {return x;}";
    printf("---Input Code---\n%s\n",code);
    printf("---Output Tokens---\n");
    analyze(code);
    return 0;
}
```

OUTPUT:

```
---Input Code---
int x=10; //start
if (x>5) {return x;}
---Output Tokens---
[KEYWORD]: int
[DELIMITER]:
[IDENTIFIER]: x
[OPERATOR]: =
[CONSTANT (Integer)]: 10
[DELIMITER]:
[OPERATOR]: /
[OPERATOR]: /
[IDENTIFIER]: start
[DELIMITER]:

[KEYWORD]: if
[DELIMITER]:
[DELIMITER]: (
[IDENTIFIER]: x
[OPERATOR]: >
[CONSTANT (Integer)]: 5
[DELIMITER]:
[DELIMITER]: {
[KEYWORD]: return
[DELIMITER]:
[IDENTIFIER]: x
[DELIMITER]: ;
[DELIMITER]: }
```

2. Write a C program to construct predictive parser for the following grammar: E→TR R→+TR/∈ T→FP T→*FP/∈ F→a/(E)

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

char stack[20];
int top = -1;

void push(char c) {
   stack[++top] = c;
}

char pop() {
   return stack[top--];
}

void error() {
   printf("Parsing not successful\n");
   exit(0);
}

void main() {
   char str[10], ch;
   int ip = 0;

   printf("Enter string ending with $: ");
   scanf("%s", str);

   push('$');
   push('E');

   while ((ch = pop()) != '$') {
      switch (ch) {
      case 'E':
         if (str[ip] == 'a' || str[ip] == '(') {
            push('R');
            push('T');
         } else error();
         break;

      case 'R':
         if (str[ip] == '+') {
            push('R');
            push('T');
            push('+');
```

```
        } else if (str[ip] != ')' && str[ip] != '$') error();
        break;

      case 'T':
        if (str[ip] == 'a' || str[ip] == '(') {
          push('P');
          push('F');
        } else error();
        break;

      case 'P':
        if (str[ip] == '*') {
          push('P');
          push('F');
          push('*');
        } else if (str[ip] != '+' && str[ip] != ')' && str[ip] != '$') error();
        break;

      case 'F':
        if (str[ip] == 'a') {
          push('a');
        } else if (str[ip] == '(') {
          push(')');
          push('E');
          push('(');
        } else error();
        break;

      case '+': case '*': case '(': case ')': case 'a':
        if (str[ip] == ch) ip++;
        break;
      }
    }

    if (str[ip] == '$')
      printf("Parsing successful\n");
    else
      printf("Parsing not successful\n");
}
```

OUTPUT - 1:

```
Enter string ending with $: a+a$
Parsing successful
```

OUTPUT - 2:

```
Enter string ending with $: a+*a$
Parsing not successful
```

3. Implementation of lexical analyzer using lex tool.

CODE:

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

%%
"/*"[^*/]*"*/" {printf("Multi-line Comment);}
"//".* {printf("Single-line Comment");}
if|else|while|do|switch|case|break|for|default {printf("Keyword");}
IF|ELSE|WHILE|DO|SWITCH|CASE|BREAK|FOR|DEFAULT {printf("Keyword");}
[A-Z a-z]+[a-z A-Z 0-9 _]* {printf("Identifier");}
[0-9]+          { printf("Integer Constant"); }
[0-9]*\.[0-9]+    { printf("Floating-point Constant"); }
[{}();,:] {printf("Delimiter");}
[ \t \n]+ {printf("Delimiter");}
%%

main()
{
   yylex();
}

/* Compilation:
```

lex lexanalysis.l

cc lex.yy.c -ll

./a.out

*/

OUTPUT:

Compilation: lex lexanalysis.l

cc lex.yy.c -ll

./a.out

Multi-line Comment: /* This is a

multi-line comment */

Identifier: int

Identifier: main

Delimiter: (

Delimiter:)

Delimiter: {

Single-line Comment: // a single-line comment

Keyword: if

Delimiter: (

Identifier: x

Unknown: =

Unknown: =

Integer Constant: 10

Delimiter:)

Identifier: x

Unknown: =

Identifier: x

Unknown: +

Integer Constant: 1

Delimiter: ;

Identifier: float

Identifier: y

Unknown: =

Floating-point Constant: 3.14

Delimiter: ;

Delimiter: }

4. Write recursive descent parser for the grammar E->E+T E->T T->T*F T->F F->(E)/id.

CODE:

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

char input[100];
int i=0;

void E();
void EPrime();
void T();
void TPrime();
void F();
void error();

int main(){
   printf("=== Recursive Descent Parser ===\n");
   printf("Grammar:\n");
   printf("E -> T E'\n");
   printf("E' -> + T E'| epsilon\n");
   printf("T -> F T'\n");
   printf("T' -> * F T'| epsilon\n");
   printf("F -> (E) | id\n\n");

   printf("Enter input string (end with $): ");
   scanf("%s",input);

   printf("\n=== Parsing Process ===\n");
   E();

   if(input[i]=='$'){
      printf("\n Parsing Successfull!\n");
   }
   else{
      printf("\n Parsing Failed!\n");
      printf("Unexpected symbol at position %d: %c\n",i,input[i]);
   }
```

```
    return 0;
}

void E(){
    printf("E -> T E'\n");
    T();
    EPrime();
}

void EPrime(){
    if(input[i]=='+'){
        printf("E' -> + T E'\n");
        i++;
        T();
        EPrime();
    }
    else{
        printf("E' -> epsilon\n");
    }
}

void T(){
    printf("T -> F T'\n");
    F();
    TPrime();
}

void TPrime(){
    if(input[i]=='*'){
        printf("T' -> * F T'\n");
        i++;
        F();
        TPrime();
    }
    else{
        printf("T' -> epsilon\n");
    }
}

void F(){
```

```
    if(input[i]=='('){
        printf("F -> ( E )\n");
        i++;
        E();
        if(input[i]==')'){
            i++;
        }
        else{
            error();
        }
    }
    else if(isalpha(input[i])||isdigit(input[i])){
        printf("F -> id (%c)\n",input[i]);
        i++;
    }
    else{
        error();
    }
}

 void error(){
    printf("\n Syntax Error at Position %d\n",i);
    printf("Unexpected symbol: %c\n",input[i]);
    exit(1);
}
```

OUTPUT:

```
=== Recursive Descent Parser ===
Grammar:
E -> T E'
E' -> + T E'| epsilon
T -> F T'
T' -> * F T'| epsilon
F -> (E) | id

Enter input string (end with $): a+b*c$

=== Parsing Process ===
E -> T E'
T -> F T'
F -> id (a)
T' -> epsilon
E' -> + T E'
T -> F T'
F -> id (b)
T' -> * F T'
F -> id (c)
T' -> epsilon
E' -> epsilon
 Parsing Successfull!
```

5. write recursive descent parser for the grammar S->(L) S->a L->L,S L->S

CODE:

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

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

char input[100];
int i=0;

void S();
void L();
void error();

int main(){
   printf("=== Recursive Descent Parser For Lists ===\n");
   printf("Grammar:\n");
   printf("S -> ( L ) | a\n");
   printf("L -> L , S | S\n\n");

   printf("Enter input string (end with $): ");
   scanf("%s", input);

   printf("\n=== Parsing Process ===\n");
   S();

   if(input[i]=='$'){
      printf("\n Parsing Successfull!\n");
   }
   else{
      printf("\n Parsing Failed!\n");
      printf("Unexpected symbol at position %d: %c\n",i,input[i]);
   }
   return 0;
}

void S(){
   if(input[i]=='('){
      printf("S -> ( L )\n");
      i++;
      L();
      if(input[i]==')'){
         i++;
      }
```

```
        else{
            error();
        }
    }
    else if(input[i]=='a'){
        printf("S -> a\n");
        i++;
    }
    else{
        error();
    }
}

void L(){
    printf("L -> S L'\n");
    S();

    while(input[i]==','){
        printf("L' -> , S L'\n");
        i++;
        S();
    }
    printf("L' -> epsilon\n");
}

void error(){
    printf("\n Syntax Error at Position %d\n",i);
    printf("Unexpected symbol: %c\n",input[i]);
    exit(1);
    }
```

OUTPUT:

```
=== Recursive Descent Parser For Lists ===
Grammar:
S -> ( L ) | a
L -> L , S | S

Enter input string (end with $): a$

=== Parsing Process ===
S -> a

 Parsing Successfull!
```

6. Write a C program to calculate first function for the grammar E->E+T E->T T->T*F T->F F->(E)/id

CODE:

```
#include<stdio.h>

#include<string.h>

#include<ctype.h>


#define MAX 10


char productions[MAX][MAX];

char first[MAX][MAX];

int n;


void findFirst(char c);

int isTerminal(char c);

void addToFirst(char c, char symbol);


int main() {

   int i;

   char choice, c;


   printf("=== FIRST SET CALCULATOR ===\n\n");

   printf("Enter number of productions: ");

   scanf("%d", &n);


   printf("Enter productions (format: E=E+T or E=@):\n");
```

```
printf("Use @ for epsilon\n");
for(i = 0; i < n; i++) {
   printf("Production %d: ", i + 1);
   scanf("%s", productions[i]);
}

// Initialize first sets
for(i = 0; i < MAX; i++)
   first[i][0] = '\0';

do {
   printf("\nEnter non-terminal to find FIRST (uppercase): ");
   scanf(" %c", &c);

   if(!isupper(c)) {
      printf("Please enter a non-terminal (uppercase letter)\n");
      continue;
   }

   findFirst(c);

   printf("\nFIRST(%c) = { ", c);
   for(i = 0; i < strlen(first[c - 'A']); i++) {
      if(i > 0) printf(", ");
      if(first[c - 'A'][i] == '@')
         printf("ε");
```

```
            else
                printf("%c", first[c - 'A'][i]);
        }
        printf(" }\n");

        printf("\nContinue? (y/n): ");
        scanf(" %c", &choice);
    } while(choice == 'y' || choice == 'Y');

    return 0;
}

void findFirst(char c) {
    int i, j, k;
    int index = c - 'A';

    // If already computed, return
    if(strlen(first[index]) > 0)
        return;

    // If terminal, add itself
    if(isTerminal(c)) {
        addToFirst(c, c);
        return;
    }
```

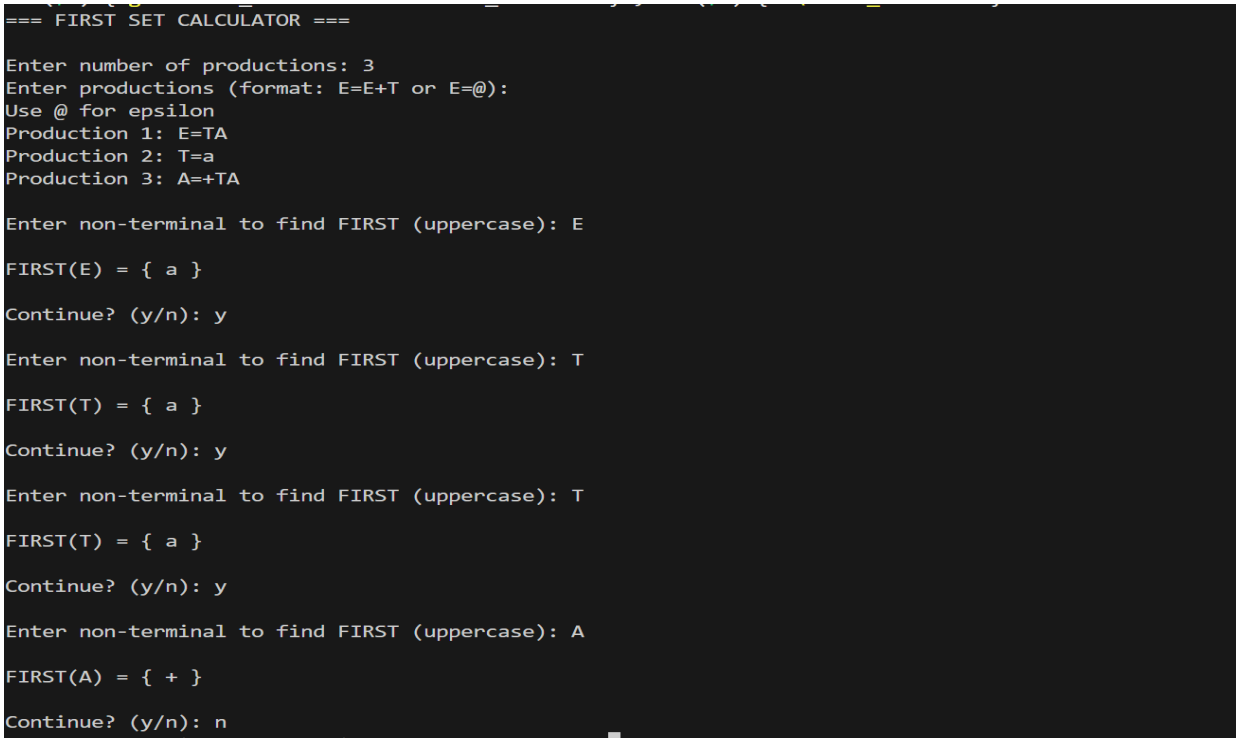
```
    // Check all productions with c on left side
    for(i = 0; i < n; i++) {
        if(productions[i][0] == c) {
            // Check right side of production
            if(productions[i][2] == '@') {  // epsilon production
                addToFirst(c, '@');
            } else if(isTerminal(productions[i][2])) {
                addToFirst(c, productions[i][2]);
            } else {
                // Non-terminal - recursively find its FIRST
                findFirst(productions[i][2]);
                for(j = 0; j < strlen(first[productions[i][2] - 'A']); j++) {
                    addToFirst(c, first[productions[i][2] - 'A'][j]);
                }
            }
        }
    }
}

int isTerminal(char c) {
    return !isupper(c);
}

void addToFirst(char c, char symbol) {
    int index = c - 'A';
    // Check if symbol already exists
```

```
    if(strchr(first[index], symbol) == NULL) {
        int len = strlen(first[index]);
        first[index][len] = symbol;
        first[index][len + 1] = '\0';
    }
}
```

OUTPUT:



7. Implementation of symbol table.

CODE:

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

#define MAX 100

struct SymbolTable
{
   char name[30];
   char type[10];
   int size;
   int address;
} st[MAX];

int count = 0;

void insert();
void display();
int search(char *);
void modify();
void delete();

int main()
{
   int choice;
   char symbol[30];
   int result;

   while (1)
   {
      printf("\n\n=== Symbol Table Operations ===\n");
      printf("1. Insert\n");
      printf("2. Display\n");
      printf("3. Search\n");
      printf("4. Modify\n");
```

```
    printf("5. Delete\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice)
    {
    case 1:
       insert();
       break;
    case 2:
       display();
       break;
    case 3:
       printf("Enter symbol to search: ");
       scanf("%s", symbol);
       result = search(symbol);
       if (result == -1)
       {
          printf("Symbol not found!\n");
       }
       else
       {
          printf("Symbol found at index %d\n", result);
       }
       break;

    case 4:
       modify();
       break;
    case 5:
       delete();
       break;
    case 6:
       exit(0);
    default:
       printf("Invalid choice!\n");
    }
}
return 0;
```

```
}

void insert()
{
   if (count >= MAX)
   {
      printf("Symbol table is full!\n");
      return;
   }

   printf("Enter symbol name: ");
   scanf("%s", st[count].name);

   if (search(st[count].name) != -1)
   {
      printf("Symbol already exists!\n");
      return;
   }

   printf("Enter type (int/float/char): ");
   scanf("%s", st[count].type);
   printf("Enter size: ");
   scanf("%d", &st[count].size);
   printf("Enter address: ");
   scanf("%d", &st[count].address);

   count++;
   printf("Symbol inserted successfully!\n");
}

void display()
{
   if (count == 0)
   {
      printf("Symbol table is empty!\n");
      return;
   }
   printf("\n%-15s %-10s %-10s %-10s\n", "Name", "Type", "Size", "Address");
   printf("-------------------------------------------------\n");
   for (int i = 0; i < count; i++)
```

```
    {
        printf("%-15s %-10s %-10d %-10d\n",
                st[i].name, st[i].type, st[i].size, st[i].address);
    }
}

int search(char *name)
{
    for (int i = 0; i < count; i++)
    {
        if (strcmp(st[i].name, name) == 0)
        {
            return i;
        }
    }
    return -1;
}

void modify()
{
    char name[30];
    int index;

    printf("Enter symbol name to modify: ");
    scanf("%s", name);

    index = search(name);
    if (index == -1)
    {
        printf("Symbol not found!\n");
        return;
    }

    printf("Enter new type: ");
    scanf("%s", st[index].type);
    printf("Enter new size: ");
    scanf("%d", &st[index].size);
    printf("Enter new address: ");
    scanf("%d", &st[index].address);
```

```
    printf("Symbol modified successfully!\n");
}

void delete()
{
    char name[30];
    int index;

    printf("Enter symbol name to delete: ");
    scanf("%s", name);

    index = search(name);
    if (index == -1)
    {
        printf("Symbol not found!\n");
        return;
    }
    for (int i = index; i < count - 1; i++)
    {
        st[i] = st[i + 1];
    }
    count--;
    printf("Symbol deleted successfully!\n");
}
```

OUTPUT:

```
=== Symbol Table Operations ===
1. Insert
2. Display
3. Search
4. Modify
5. Delete
6. Exit
Enter your choice: 1
Enter symbol name: x
Enter type (int/float/char): int
Enter size: 4
Enter address: 100
Symbol inserted successfully!


=== Symbol Table Operations ===
1. Insert
2. Display
3. Search
4. Modify
5. Delete
6. Exit
Enter your choice: 1
Enter symbol name: total
Enter type (int/float/char): float
Enter size: 8
Enter address: 200
Symbol inserted successfully!
```

```
=== Symbol Table Operations ===
1. Insert
2. Display
3. Search
4. Modify
5. Delete
6. Exit
Enter your choice: 2

Name            Type        Size        Address
-----------------------------------------------------
x               int         4           100
total           float       8           200


=== Symbol Table Operations ===
1. Insert
2. Display
3. Search
4. Modify
5. Delete
6. Exit
Enter your choice: 3
Enter symbol to search: total
Symbol found at index 1


=== Symbol Table Operations ===
1. Insert
2. Display
3. Search
4. Modify
5. Delete
6. Exit
Enter your choice: 4
Enter symbol name to modify: x
Enter new type: char
Enter new size: 1
Enter new address: 300
Symbol modified successfully!


=== Symbol Table Operations ===
1. Insert
2. Display
3. Search
4. Modify
5. Delete
6. Exit
Enter your choice: 2

Name            Type        Size        Address
-----------------------------------------------------
x               char        1           300
total           float       8           200


=== Symbol Table Operations ===
1. Insert
2. Display
3. Search
4. Modify
5. Delete
6. Exit
Enter your choice: 5
Enter symbol name to delete: total
Symbol deleted successfully!

=== Symbol Table Operations ===
1. Insert
2. Display
3. Search
4. Modify
5. Delete
6. Exit
Enter your choice: 6
```

8. Implement type checking

CODE:

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

#define MAX 50

typedef struct
{
   char name[30];
   char type[10];
} Symbol;

Symbol symbolTable[MAX];
int symbolCount = 0;

void addSymbol(char *name,char *type);
char* getType(char *name);
int typeCheck(char *type1,char *op, char *type2);
char* resultType(char *type1,char *type2);

int main(){
   char var1[30], var2[30], op[5], result[10];
   int choice;
   while(1){
      printf("\n=== Type Checking System ===\n");
      printf("1. Declare Variable\n");
      printf("2. Check Expression Type\n");
      printf("3. Display Symbol Table\n");
      printf("4. Exit\n");
      printf("Enter Choice: ");
      scanf("%d",&choice);

      switch(choice){
```

```
case 1:
   printf("Enter variable name: ");
   scanf("%s",var1);
   printf("Enter type (int/float/char): ");
   scanf("%s",result);
   addSymbol(var1,result);
   printf("Variable declared successfully!\n");
   break;

case 2:
   printf("Enter expression (var1 op var2): ");
   scanf("%s %s %s",var1, op, var2);
   char *type1=getType(var1);
   char *type2=getType(var2);
   if(type1==NULL||type2==NULL){
      printf("Type Error: Undeclared variable!\n");
      break;
   }
   if(typeCheck(type1,op,type2)){
      char *resType=resultType(type1,type2);
      printf("Type Checking PASSED!\n");
      printf("Result Type: %s\n",resType);
   }
   else{
      printf("Type Error: Incompatible types for operation!\n");
   }
   break;
case 3:
   printf("\n=== Symbol Table ===\n");
   printf("%-15s %-10s\n","Name","Type");
   printf("--------------------------\n");
   for(int i=0;i<symbolCount;i++){
      printf("%-15s %-10s\n",symbolTable[i].name,symbolTable[i].type);
   }
   break;
```

```
            case 4:
                exit(0);

            default:
                printf("Invalid Choice!\n");

        }
    }
    return 0;
}

void addSymbol(char *name,char *type){
    strcpy(symbolTable[symbolCount].name,name);
    strcpy(symbolTable[symbolCount].type,type);
    symbolCount++;
}

char* getType(char *name){
    for(int i=0;i<symbolCount;i++){
        if(strcmp(symbolTable[i].name,name)==0){
            return symbolTable[i].type;
        }
    }
    return NULL;
}

int typeCheck(char *type1,char *op,char *type2){
    if(strcmp(op,"+")==0||strcmp(op,"-
")==0||strcmp(op,"*")==0||strcmp(op,"/")==0){

if((strcmp(type1,"int")==0||strcmp(type1,"float")==0)&&(strcmp(type2,"int")==0|
|strcmp(type2,"float")==0)){
            return 1;
        }
```

```
    }
    if(strcmp(type1,type2)==0){
        return 1;
    }
    return 0;
}

char* resultType(char *type1,char *type2){
    if(strcmp(type1,"float")==0||strcmp(type2,"float")==0){
        return "float";
    }
    return type1;
}
```

OUTPUT:

```
=== Type Checking System ===
1. Declare Variable
2. Check Expression Type
3. Display Symbol Table
4. Exit
Enter Choice: 1
Enter variable name: x
Enter type (int/float/char): int
Variable declared successfully!

=== Type Checking System ===
1. Declare Variable
2. Check Expression Type
3. Display Symbol Table
4. Exit
Enter Choice: 1
Enter variable name: y
Enter type (int/float/char): float
Variable declared successfully!

=== Type Checking System ===
1. Declare Variable
2. Check Expression Type
3. Display Symbol Table
4. Exit
Enter Choice: 2
Enter expression (var1 op var2): x + y
Type Checking PASSED!
Result Type: float
```

```
=== Type Checking System ===
1. Declare Variable
2. Check Expression Type
3. Display Symbol Table
4. Exit
Enter Choice: 3

=== Symbol Table ===
Name            Type
--------------------------
x               int
y               float

=== Type Checking System ===
1. Declare Variable
2. Check Expression Type
3. Display Symbol Table
4. Exit
Enter Choice: 4
```

9. Implementation of storage allocation strategies (heap, stack,static)

CODE:

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

#define STACK_SIZE 1000

typedef struct
{
   char name[30];
   char type[10];
   int size;
   int offset;
} StackFrame;

StackFrame stack[100];
int frameCount = 0;
int stackPointer = 0;

void allocate(char *name, char *type, int size);
void deallocate();
void display();
int getSize(char *type);

int main()
{
   int choice, size;
   char name[30], type[10];
   while (1)
   {
      printf("\n=== Stack Storage Allocation ===\n");
      printf("Stack Size: %d bytes\n", STACK_SIZE);
      printf("Used: %d bytes\n", stackPointer);
      printf("Free: %d bytes\n\n", STACK_SIZE - stackPointer);

      printf("1. Allocate Variable\n");
      printf("2. Deallocate (Pop)\n");
      printf("3. Display Stack\n");
```

```
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("Enter variable name: ");
                scanf("%s", name);
                printf("Enter type (int/float/char/double):");
                scanf("%s", type);
                size = getSize(type);
                allocate(name, type, size);
                break;
            case 2:
                deallocate();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
}

void allocate(char *name, char *type, int size)
{
    if (stackPointer + size > STACK_SIZE)
    {
        printf("Stack Overflow! Cannot allocate!\n");
        return;
    }
    strcpy(stack[frameCount].name, name);
    strcpy(stack[frameCount].type, type);
    stack[frameCount].size = size;
    stack[frameCount].offset = stackPointer;
```

```
    stackPointer += size;
    frameCount++;

    printf("Allocated %d bytes for %s at offset %d\n", size, name, stackPointer - size);
}

void deallocate()
{
    if (frameCount == 0)
    {
        printf("Stack Underflow! Stack is empty.\n");
        return;
    }
    frameCount--;
    stackPointer -= stack[frameCount].size;
    printf("Deallocated '%s' (%d bytes)\n", stack[frameCount].name,
stack[frameCount].size);
}

void display()
{
    if (frameCount == 0)
    {
        printf("Stack is Empty!\n");
        return;
    }

    printf("\n=== Stack Frame Contents ===\n");
    printf("%-15s %-10s %-10s %-10s\n", "Variable", "Type", "Size", "Offset");
    printf("------------------------------------------------\n");
    for (int i = 0; i < frameCount; i++)
    {
        printf("%-15s %-10s %-10d %-10d\n",
              stack[i].name, stack[i].type, stack[i].size, stack[i].offset);
    }
    printf("\nStack Pointer: %d\n", stackPointer);
}

int getSize(char *type)
{
```

```
    if (strcmp(type, "int") == 0)
    {
        return 4;
    }
    else if (strcmp(type, "float") == 0)
    {
        return 4;
    }
    else if (strcmp(type, "char") == 0)
    {
        return 1;
    }
    else if (strcmp(type, "double") == 0)
    {
        return 8;
    }
    return 4;
}
```

OUTPUT:

```
=== Stack Storage Allocation ===
Stack Size: 1000 bytes
Used: 0 bytes
Free: 1000 bytes

1. Allocate Variable
2. Deallocate (Pop)
3. Display Stack
4. Exit
Enter choice: 1
Enter variable name: x
Enter type (int/float/char/double):int
Allocated 4 bytes for x at offset 0

=== Stack Storage Allocation ===
Stack Size: 1000 bytes
Used: 4 bytes
Free: 996 bytes

1. Allocate Variable
2. Deallocate (Pop)
3. Display Stack
4. Exit
Enter choice: 1
Enter variable name: y
Enter type (int/float/char/double):double
Allocated 8 bytes for y at offset 4

=== Stack Storage Allocation ===
Stack Size: 1000 bytes
Used: 12 bytes
Free: 988 bytes
```

```
1. Allocate Variable
2. Deallocate (Pop)
3. Display Stack
4. Exit
Enter choice: 1
Enter variable name: z
Enter type (int/float/char/double):char
Allocated 1 bytes for z at offset 12

=== Stack Storage Allocation ===
Stack Size: 1000 bytes
Used: 13 bytes
Free: 987 bytes

1. Allocate Variable
2. Deallocate (Pop)
3. Display Stack
4. Exit
Enter choice: 3

=== Stack Frame Contents ===
Variable         Type         Size         Offset
--------------------------------------------------
x                int          4            0
y                double       8            4
z                char         1            12

Stack Pointer: 13

=== Stack Storage Allocation ===
Stack Size: 1000 bytes
Used: 13 bytes
Free: 987 bytes
```

```
1. Allocate Variable
2. Deallocate (Pop)
3. Display Stack
4. Exit
Enter choice: 2
Deallocated 'z' (1 bytes)

=== Stack Storage Allocation ===
Stack Size: 1000 bytes
Used: 12 bytes
Free: 988 bytes

1. Allocate Variable
2. Deallocate (Pop)
3. Display Stack
4. Exit
Enter choice: 3

=== Stack Frame Contents ===
Variable         Type         Size         Offset
--------------------------------------------------
x                int          4            0
y                double       8            4

Stack Pointer: 12

=== Stack Storage Allocation ===
Stack Size: 1000 bytes
Used: 12 bytes
Free: 988 bytes

1. Allocate Variable
2. Deallocate (Pop)
3. Display Stack
4. Exit
```

10. Write a Lex specification to recognize +ve integers,reals and -ve integers,reals.

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

%%
"+"?[0-9]+ {printf("%s:positive integers",yytext);}
-[0-9]+ {printf("%s:negative integers",yytext);}
-[0-9]+\.[0-9]+ {printf("%s:negative real numbers",yytext);}
"+"?[0-9]+\.[0-9]+ {printf("%s:positive real numbers",yytext);}
%%

main()
{
   yylex();
}
```

OUTPUT:

Compilation: lex noformat.l

cc lex.yy.c -ll

```
./a.out
24
positive integer
+24.12
positive real number
-24
negative integer
-24.12
negative real number
```

11. Write a Lex specification for converting real numbers to integers.

```
%{
int i,j;
#include<stdio.h>
%}

%%
[0-9]*\.[0-9]+ {
   for(i=0;i<10;i++)
   {
      if(yytext[i]=='.')
         for(j=0;j<=i-1;j++)
         {
            printf("%c",yytext[j]);
         }
   }
   exit(0);
}
%%

main()
{
   yylex();
}

/* Compilation:
  lex realtoint.l
  cc lex.yy.c -ll
  ./a.out
*/
```

OUTPUT:

Compilation: lex realtoint.l
cc lex.yy.c -ll
./a.out
24.12
12

12. Write a Lex specification to print the number of days in a month using a procedure

```
%{
#include<stdio.h>
int year;
%}

%%
jan|mar|may|july|aug|oct|dec {printf("31 days in %s",yylex);}
april|june|sep|nov {printf("30 days in %s",yylex);}
feb {leap();}
[a-z A-Z]* {printf("invalid");}
%%

main()
{
   yylex();
}

leap()
{
   printf("enter year");
   scanf("%d", &year);
   if((year%4==0&&year%100!=0)||(year%400==0))
      printf("29 days");
   else
      printf("28 days");
}
```

Compilation: lex daysinamonth.l

```
cc lex.yy.c -ll
./a.out
jan
31 days
june
30 days
feb
enter year
1984
29 days
```

13. Write a lex program to count the number of words and number of lines in a given file or program.

```
//wordcount.l

%{
#include<stdio.h>
int lineCount=0, wordCount=0, charCount=0;
%}

%%
\n {lineCount++; charCount++}
[ \t]+ {charCount+=yyleng;}
[^ \t\n]+ {wordCount++; charCount+=yyleng;}
%%

int yywrap(){
   return 1;
}

main(){
   printf("Enter text: \n" );
   yylex();

   printf("\n== FILE STATISTICS ==\n);
   printf("Number of lines: %d\n",lineCount);
   printf("Number of words: %d\n",wordCount);
   printf("Number of characters: %d\n",charCount);
}
```

Compilation:

```
lex wordcount.l
cc lex.yy.c -ll -o wordcount
./wordcount
```

OUPUT:

```
$ ./wordcount
Enter text:
Hello world
This is a test.
<Ctrl-D>

== FILE STATISTICS ==
Number of lines: 2
Number of words: 6
Number of characters: 28
```

14. Write a Lex specification to retrieve comments.

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

%%
[/][/] [a-z A-Z 0-9]* {printf("%s",yytext);}
[a-z A-Z 0-9]* {printf(" ");}
[/][*][a-z A-Z 0-9]*[*][/] {printf("%s",yytext);}
%%

main()
{
   yylex();
}

/* Compilation:
  lex comments.l
  cc lex.yy.l -ll
  ./a.out
*/
```

OUTPUT:

Compilation: lex comments.l
cc lex.yy.c -ll
./a.out
Hello //world
world
hai /*friend*/
friend