**EXP 5:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX\_IDENTIFIER\_LENGTH 100

int isValidIdentifier(const char \*identifier) {

if (!isalpha(identifier[0]) && identifier[0] != '\_') {

return 0;

}

for (int i = 1; i < strlen(identifier); i++) {

if (!isalnum(identifier[i]) && identifier[i] != '\_') {

return 0;

}

}

return 1;

}

int main() {

char identifier[MAX\_IDENTIFIER\_LENGTH];

printf("Enter an identifier: ");

scanf("%s", identifier);

if (isValidIdentifier(identifier)) {

printf("'%s' is a valid identifier.\n", identifier);

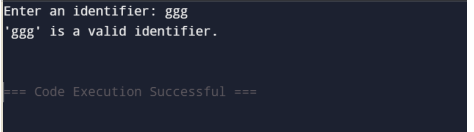
} else {

printf("'%s' is not a valid identifier.\n", identifier);

}

return 0;

}



**EXP 6:**

#include <stdio.h>

#include <string.h>

#define MAX 100

void eliminateLeftRecursion(char nonTerminal, char \*production) {

char alpha[MAX], beta[MAX];

int i = 0, j = 0, k = 0;

if (production[0] == nonTerminal) {

i = 1;

while (production[i] != '|' && production[i] != '\0') {

alpha[j++] = production[i++];

}

alpha[j] = '\0';

i++;

while (production[i] != '\0') {

beta[k++] = production[i++];

}

beta[k] = '\0';

printf("Grammar after eliminating left recursion:\n");

printf("%c -> %s%c'\n", nonTerminal, beta, nonTerminal + 1);

printf("%c' -> %s%c' | ε\n", nonTerminal + 1, alpha, nonTerminal + 1);

} else {

printf("The production does not have left recursion.\n");

}

}

int main() {

char nonTerminal;

char production[MAX];

printf("Enter the non-terminal (e.g., A): ");

scanf(" %c", &nonTerminal);

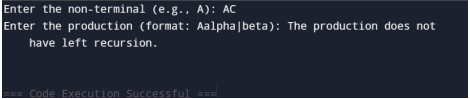
printf("Enter the production (format: Aalpha|beta): ");

scanf(" %s", production);

eliminateLeftRecursion(nonTerminal, production);

return 0;

}



**EXP 7:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX 10

void leftFactor(char nonTerminal, char \*productions[], int count) {

char prefix[MAX], newSymbol, newProduction[MAX][MAX];

int i, j, k, prefixLength = 0, flag = 1;

for (i = 0; i < strlen(productions[0]); i++) {

char ch = productions[0][i];

for (j = 1; j < count; j++) {

if (productions[j][i] != ch) {

flag = 0;

break;

}

}

if (!flag) break;

prefix[prefixLength++] = ch;

}

prefix[prefixLength] = '\0';

if (prefixLength == 0) {

printf("%c -> ", nonTerminal);

for (i = 0; i < count; i++) {

printf("%s", productions[i]);

if (i < count - 1) printf(" | ");

}

printf("\n");

return;

}

newSymbol = 'X';

printf("%c -> %s%c\n", nonTerminal, prefix, newSymbol);

printf("%c -> ", newSymbol);

for (i = 0; i < count; i++) {

if (strlen(productions[i]) == prefixLength) {

printf("ε"); // If only prefix, use epsilon

} else {

printf("%s", productions[i] + prefixLength);

}

if (i < count - 1) printf(" | ");

}

printf("\n");

}

int main() {

char nonTerminal;

int i, count;

char \*productions[MAX];

printf("Enter the non-terminal: ");

scanf(" %c", &nonTerminal);

printf("Enter the number of productions: ");

scanf("%d", &count);

for (i = 0; i < count; i++) {

productions[i] = (char \*)malloc(MAX \* sizeof(char));

printf("Enter production %d: ", i + 1);

scanf("%s", productions[i]);

}

printf("\nGrammar after left factoring:\n");

leftFactor(nonTerminal, productions, count);

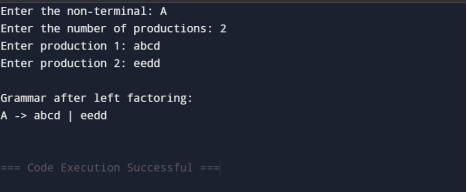
for (i = 0; i < count; i++) {

free(productions[i]);

}

return 0;

}



**EXP 8:**

#include <stdio.h>

#include <string.h>

#define MAX 100

struct Symbol {

char name[50];

char type[20];

int address;

} table[MAX];

int count = 0;

void insert() {

if (count >= MAX) {

printf("Symbol Table is Full!\n");

return;

}

char name[50], type[20];

int address, i;

printf("Enter Symbol Name: ");

scanf("%s", name);

printf("Enter Type: ");

scanf("%s", type);

printf("Enter Address: ");

scanf("%d", &address);

for (i = 0; i < count; i++) {

if (strcmp(table[i].name, name) == 0) {

printf("Error: Symbol already exists!\n");

return;

}

}

strcpy(table[count].name, name);

strcpy(table[count].type, type);

table[count].address = address;

count++;

printf("Symbol Inserted Successfully!\n");

}

void search() {

char name[50];

printf("Enter Symbol Name to Search: ");

scanf("%s", name);

for (int i = 0; i < count; i++) {

if (strcmp(table[i].name, name) == 0) {

printf("Symbol Found: Name: %s, Type: %s, Address: %d\n",

table[i].name, table[i].type, table[i].address);

return;

}

}

printf("Symbol Not Found!\n");

}

void display() {

if (count == 0) {

printf("Symbol Table is Empty!\n");

return;

}

printf("\nSymbol Table:\n");

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

printf("Index\tName\t\tType\t\tAddress\n");

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

for (int i = 0; i < count; i++) {

printf("%d\t%-10s\t%-10s\t%d\n", i + 1, table[i].name, table[i].type, table[i].address);

}

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

}

int main() {

int choice;

while (1) {

printf("\nSymbol Table Operations:\n");

printf("1. Insert Symbol\n");

printf("2. Search Symbol\n");

printf("3. Display Symbol Table\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: insert(); break;

case 2: search(); break;

case 3: display(); break;

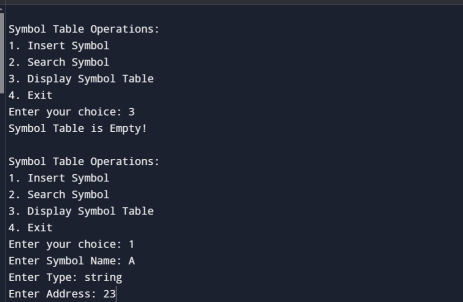
case 4: return 0;

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

}

}

}



**EXP 9:**

#include <stdio.h>

#include <string.h>

const char \*subjects[] = {"I", "You", "We"};

const char \*verbs[] = {"eat", "play", "read"};

const char \*objects[] = {"food", "cricket", "book"};

#define SUBJECTS\_SIZE 3

#define VERBS\_SIZE 3

#define OBJECTS\_SIZE 3

int isInList(const char \*word, const char \*list[], int size) {

for (int i = 0; i < size; i++) {

if (strcmp(word, list[i]) == 0)

return 1;

}

return 0;

}

void checkGrammar(char \*sentence) {

char \*subject, \*verb, \*object;

subject = strtok(sentence, " ");

verb = strtok(NULL, " ");

object = strtok(NULL, " ");

if (subject == NULL || verb == NULL || object == NULL || strtok(NULL, " ") != NULL) {

printf("Invalid Sentence!\n");

return;

}

if (isInList(subject, subjects, SUBJECTS\_SIZE) &&

isInList(verb, verbs, VERBS\_SIZE) &&

isInList(object, objects, OBJECTS\_SIZE)) {

printf("Valid Sentence!\n");

} else {

printf("Invalid Sentence!\n");

}

}

int main() {

char sentence[100];

printf("Enter a sentence: ");

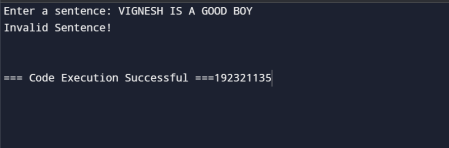
fgets(sentence, sizeof(sentence), stdin);

sentence[strcspn(sentence, "\n")] = '\0';

checkGrammar(sentence);

return 0;

}



**EXP 10:**

#include <stdio.h>

#include <math.h>

int main() {

int a = 3, b = 6, c = 5, d = 4, e = 3, f = 7;

double result;

printf("Given Expression: 3 + 6 \* (5 + 4) / 3 - 7\n\n");

printf("Step 1 (Parentheses first): (5 + 4) = 9\n");

int step1 = c + d;

printf("Step 2 (Multiplication first): 6 \* 9 = 54\n");

int step2 = b \* step1;

printf("Step 3 (Division next): 54 / 3 = 18\n");

int step3 = step2 / e;

printf("Step 4 (Addition next): 3 + 18 = 21\n");

int step4 = a + step3;

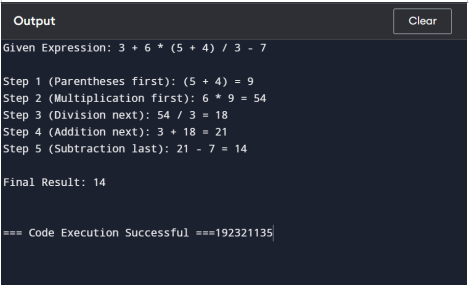
printf("Step 5 (Subtraction last): 21 - 7 = 14\n");

result = step4 - f;

printf("\nFinal Result: %d\n", (int)result);

return 0;

}



**EXP 11:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

char input[100];

int pos = 0;

void E();

void T();

void F();

void match(char expected) {

if (input[pos] == expected)

pos++;

else {

printf("Error: Unexpected character '%c'\n", input[pos]);

exit(1);

}

}

void E() {

T();

if (input[pos] == '+') {

match('+');

E();

}

}

void T() {

F();

if (input[pos] == '\*') {

match('\*');

T();

}

}

void F() {

if (isalpha(input[pos])) {

match(input[pos]);

} else if (input[pos] == '(') {

match('(');

E();

match(')');

} else {

printf("Error: Invalid character '%c'\n", input[pos]);

exit(1);

}

}

int main() {

printf("Enter an expression: ");

scanf("%s", input);

E();

if (input[pos] == '\0')

printf("Valid Expression!\n");

else

printf("Error: Unexpected characters after parsing!\n");

return 0



**EXP 12:**

#include <stdio.h>

#include <string.h>

void generateTAC(char expression[]) {

printf("\nGenerating Three Address Code for: %s\n", expression);

printf("\nStep-by-Step TAC Representation:\n");

printf("t1 = c \* d\n");

printf("t2 = e / f\n");

printf("t3 = b + t1\n");

printf("t4 = t3 - t2\n");

printf("a = t4\n");

}

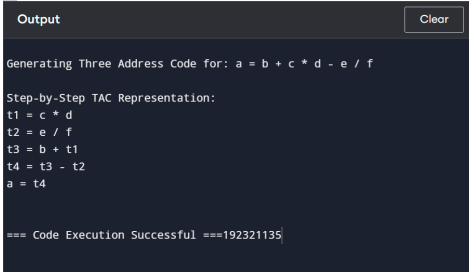
int main() {

char expression[50] = "a = b + c \* d - e / f";

generateTAC(expression);

return 0;

}



**EXP 13:**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

int main() {

FILE \*file;

char filename[100], ch;

int characters = 0, words = 0, lines = 0;

int inWord = 0;

printf("Enter the filename: ");

scanf("%s", filename);

file = fopen(filename, "r");

if (file == NULL) {

printf("Error opening file.\n");

return 1;

}

while ((ch = fgetc(file)) != EOF) {

characters++;

if (ch == '\n') {

lines++;

}

if (isspace(ch) || ch == '\n' || ch == '\t') {

inWord = 0;

} else if (inWord == 0) {

inWord = 1;

words++;

}

}

fclose(file);

printf("\nFile Analysis Results:\n");

printf("Total Characters: %d\n", characters);

printf("Total Words: %d\n", words);

printf("Total Lines: %d\n", lines);

return 0;

}