Program no:11

Implement a C program to perform symbol table operations.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SYMBOLS 100

#define MAX\_NAME\_LENGTH 50

typedef struct {

char name[MAX\_NAME\_LENGTH];

int value;

} Symbol;

typedef struct {

Symbol symbols[MAX\_SYMBOLS];

int count;

} SymbolTable;

void initSymbolTable(SymbolTable \*table) {

table->count = 0;

}

int addSymbol(SymbolTable \*table, const char \*name, int value) {

if (table->count >= MAX\_SYMBOLS) {

return -1; // Table is full

}

strcpy(table->symbols[table->count].name, name);

table->symbols[table->count].value = value;

table->count++;

return 0; // Success

}

int getSymbolValue(SymbolTable \*table, const char \*name) {

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

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

return table->symbols[i].value;

}

}

return -1; // Symbol not found

}

void printSymbolTable(SymbolTable \*table) {

printf("Symbol Table:\n");

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

printf("Name: %s, Value: %d\n", table->symbols[i].name, table->symbols[i].value);

}

}

int main() {

SymbolTable table;

initSymbolTable(&table);

addSymbol(&table, "x", 10);

addSymbol(&table, "y", 20);

addSymbol(&table, "z", 30);

printSymbolTable(&table);

int value = getSymbolValue(&table, "y");

if (value != -1) {

printf("Value of y: %d\n", value);

} else {

printf("Symbol not found.\n");

}

return 0;

}

Output:

Symbol Table:

Name: x, Value: 10

Name: y, Value: 20

Name: z, Value: 30

Value of y: 20

Program no:12

Write a C program to construct recursive descent parsing for the given grammar

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

char input[100];

int current\_char\_index = 0;

void expr();

void term();

void factor();

void match(char expected) {

if (input[current\_char\_index] == expected) {

current\_char\_index++;

} else {

printf("Error: Expected '%c' but found '%c' at %d\n", expected, input[current\_char\_index], current\_char\_index + 1);

exit(1);

}

}

void expr() {

term();

while (input[current\_char\_index] == '+' || input[current\_char\_index] == '-') {

match(input[current\_char\_index]);

term();

}

}

void term() {

factor();

while (input[current\_char\_index] == '\*' || input[current\_char\_index] == '/') {

match(input[current\_char\_index]);

factor();

}

}

void factor() {

if (isdigit(input[current\_char\_index])) {

while(isdigit(input[current\_char\_index])){

match(input[current\_char\_index]);

}

} else if (input[current\_char\_index] == '(') {

match('(');

expr();

match(')');

} else {

printf("Error: Invalid input at %d\n", current\_char\_index + 1);

exit(1);

}

}

int main() {

printf("Enter an expression: ");

scanf("%99s", input); // Prevent buffer overflow

expr();

if (input[current\_char\_index] == '\0') {

printf("Valid expression.\n");

} else {

printf("Error: Extra characters at the end of input.\n");

}

return 0;

}

Output:

Enter an expression: 10+2\*5

Valid expression.