21BDS0340

Abhinav Dinesh Srivatsa

Compiler Design Lab

Assignment. – II

**Exercise 1**

**Question 1:**

Aim:

Write a C/C++ program to find tokens in the following line: if (a == b) c = a;

Program:

#include <iostream>

#include <string>

using namespace std;

bool checkKeyword(string ch)

{

string keywords[32] = {"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"};

for (int x = 0; x < 32; x++)

if (ch == keywords[x])

return true;

return false;

}

bool checkReal(string ch)

{

if (ch == "")

return false;

if (ch[0] == '.')

return false;

for (int x = 0; x < ch.length(); x++)

if (!(48 <= ch[x] && ch[x] <= 57))

return false;

return true;

}

bool checkOperator(char ch)

{

if (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '=' ||

ch == '>' || ch == '<' || ch == '!' || ch == '|' || ch == '&')

return true;

return false;

}

bool checkSpecial(char ch)

{

if (ch == ',' || ch == ';' || ch == '(' || ch == ')' ||

ch == '{' || ch == '}' || ch == '[' || ch == ']' ||

ch == ' ' || ch == '\n')

return true;

return false;

}

char \*stringifyCode()

{

return NULL;

}

int main()

{

string code = "if (a == b) c = a;";

string str = "";

for (int x = 0; x < code.length(); x++)

{

char c = code[x];

if (checkOperator(c))

{

cout << c << " is an operator\n";

if (checkReal(str))

{

cout << str << " is a constant\n";

str = "";

continue;

}

if (checkKeyword(str))

{

cout << str << " is a keyword\n";

str = "";

continue;

}

if (!(str == ""))

cout << str << " is a variable\n";

str = "";

continue;

}

if (checkSpecial(c))

{

if (!(c == ' ' || c == '\n'))

cout << c << " is a special\n";

if (checkReal(str))

{

cout << str << " is a constant\n";

str = "";

continue;

}

if (checkKeyword(str))

{

cout << str << " is a keyword\n";

str = "";

continue;

}

if (!(str == ""))

cout << str << " is a variable\n";

str = "";

continue;

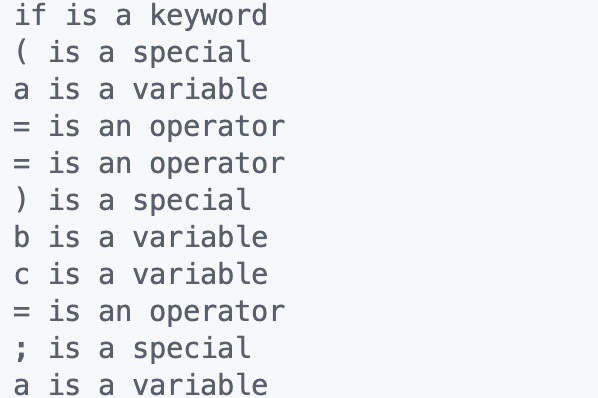
}

str.push\_back(c);

}

}

**Output:**



**Question 2:**

Aim:

Write a C/C++ program to find whether a given string is an identifier or not

Program:

#include <iostream>

#include <string>

using namespace std;

bool checkKeyword(string ch)

{

string keywords[32] = {"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"};

for (int x = 0; x < 32; x++)

if (ch == keywords[x])

return true;

return false;

}

bool checkReal(string ch)

{

if (ch == "")

return false;

if (ch[0] == '.')

return false;

for (int x = 0; x < ch.length(); x++)

if (!(48 <= ch[x] && ch[x] <= 57))

return false;

return true;

}

bool checkOperator(char ch)

{

if (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '=' ||

ch == '>' || ch == '<' || ch == '!' || ch == '|' || ch == '&')

return true;

return false;

}

bool checkSpecial(char ch)

{

if (ch == ',' || ch == ';' || ch == '(' || ch == ')' ||

ch == '{' || ch == '}' || ch == '[' || ch == ']' ||

ch == ' ' || ch == '\n')

return true;

return false;

}

int main()

{

string code = "if (a == b) c = a;";

string str = "";

for (int x = 0; x < code.length(); x++)

{

char c = code[x];

if (checkOperator(c))

{

if (checkReal(str))

{

str = "";

continue;

}

if (checkKeyword(str))

{

str = "";

continue;

}

if (!(str == ""))

cout << str << " is an identifier\n";

str = "";

continue;

}

if (checkSpecial(c))

{

if (!(c == ' ' || c == '\n'))

if (checkReal(str))

{

str = "";

continue;

}

if (checkKeyword(str))

{

str = "";

continue;

}

if (!(str == ""))

cout << str << " is an identifier\n";

str = "";

continue;

}

str.push\_back(c);

}

}

**Output:**



**Question 3:**

Aim:

Write a C/C++ program to scan and count the number of characters and words in a line

Program:

#include <iostream>

#include <string>

using namespace std;

bool checkLetter(char c)

{

if (c >= 'a' && c <= 'z')

return true;

if (c >= 'A' && c <= 'Z')

return true;

return false;

}

int main()

{

string code = "if (a == b) c = a;";

string str = "";

int characters = code.length(), words = 0;

for (int x = 0; x < code.length(); x++)

{

if (x == 0)

{

words++;

continue;

}

if (checkLetter(code[x]) && !checkLetter(code[x - 1]))

words++;

}

cout << "Characters: " << characters << "\nWords: " << words << "\n";

}

Output:

A picture containing text, font, white, calligraphy

Description automatically generated

**Question 4:**

Aim:

Write a C/C++ program to find whether a given string is a keyword or not

Program:

#include <iostream>

#include <string>

using namespace std;

bool checkKeyword(string ch)

{

string keywords[32] = {"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"};

for (int x = 0; x < 32; x++)

if (ch == keywords[x])

return true;

return false;

}

int main()

{

string input;

cin >> input;

if (checkKeyword(input))

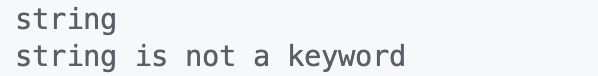
cout << input << " is a keyword\n";

else

cout << input << " is not a keyword\n";

}

Output:





**Exercise 2**

**Question 1:**

Aim:

Write a C/C++ program to convert NFA to DFA

Program:

#include <stdio.h>

int main()

{

int nfa[5][2];

nfa[1][1] = 12;

nfa[1][2] = 1;

nfa[2][1] = 0;

nfa[2][2] = 3;

nfa[3][1] = 0;

nfa[3][2] = 4;

nfa[4][1] = 0;

nfa[4][2] = 0;

int dfa[10][2];

int dstate[10];

int i = 1, n, j, k, flag = 0, m, q, r;

dstate[i++] = 1;

n = i;

dfa[1][1] = nfa[1][1];

dfa[1][2] = nfa[1][2];

printf("\nf(%d,a)=%d", dstate[1], dfa[1][1]);

printf("\nf(%d,b)=%d", dstate[1], dfa[1][2]);

for (j = 1; j < n; j++)

{

if (dfa[1][1] != dstate[j])

flag++;

}

if (flag == n - 1)

{

dstate[i++] = dfa[1][1];

n++;

}

flag = 0;

for (j = 1; j < n; j++)

{

if (dfa[1][2] != dstate[j])

flag++;

}

if (flag == n - 1)

{

dstate[i++] = dfa[1][2];

n++;

}

k = 2;

while (dstate[k] != 0)

{

m = dstate[k];

if (m > 10)

{

q = m / 10;

r = m % 10;

}

if (nfa[r][1] != 0)

dfa[k][1] = nfa[q][1] \* 10 + nfa[r][1];

else

dfa[k][1] = nfa[q][1];

if (nfa[r][2] != 0)

dfa[k][2] = nfa[q][2] \* 10 + nfa[r][2];

else

dfa[k][2] = nfa[q][2];

printf("\nf(%d,a)=%d", dstate[k], dfa[k][1]);

printf("\nf(%d,b)=%d", dstate[k], dfa[k][2]);

flag = 0;

for (j = 1; j < n; j++)

{

if (dfa[k][1] != dstate[j])

flag++;

}

if (flag == n - 1)

{

dstate[i++] = dfa[k][1];

n++;

}

flag = 0;

for (j = 1; j < n; j++)

{

if (dfa[k][2] != dstate[j])

flag++;

}

if (flag == n - 1)

{

dstate[i++] = dfa[k][2];

n++;

}

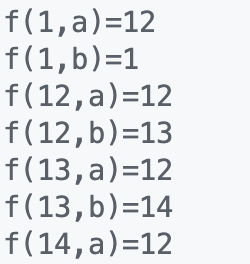
k++;

}

return 0;

}

Output:



**Question 2:**

Aim:

Write a C/C++ program to implement a symbol table

Program:

#include <iostream>

using namespace std;

const int MAX = 100;

class Node

{

string identifier, scope, type;

int lineNo;

Node \*next;

public:

Node()

{

next = NULL;

}

Node(string key, string value, string type, int lineNo)

{

this->identifier = key;

this->scope = value;

this->type = type;

this->lineNo = lineNo;

next = NULL;

}

void print()

{

cout << "Identifier's Name:" << identifier

<< "\nType:" << type

<< "\nScope: " << scope

<< "\nLine Number: " << lineNo << endl;

}

friend class SymbolTable;

};

class SymbolTable

{

Node \*head[MAX];

public:

SymbolTable()

{

for (int i = 0; i < MAX; i++)

head[i] = NULL;

}

int hashf(string id);

bool insert(string id, string scope,

string Type, int lineno);

string find(string id);

bool deleteRecord(string id);

bool modify(string id, string scope,

string Type, int lineno);

};

bool SymbolTable::modify(string id, string s,

string t, int l)

{

int index = hashf(id);

Node \*start = head[index];

if (start == NULL)

return "-1";

while (start != NULL)

{

if (start->identifier == id)

{

start->scope = s;

start->type = t;

start->lineNo = l;

return true;

}

start = start->next;

}

return false;

}

bool SymbolTable::deleteRecord(string id)

{

int index = hashf(id);

Node \*tmp = head[index];

Node \*par = head[index];

if (tmp == NULL)

{

return false;

}

if (tmp->identifier == id && tmp->next == NULL)

{

tmp->next = NULL;

delete tmp;

return true;

}

while (tmp->identifier != id && tmp->next != NULL)

{

par = tmp;

tmp = tmp->next;

}

if (tmp->identifier == id && tmp->next != NULL)

{

par->next = tmp->next;

tmp->next = NULL;

delete tmp;

return true;

}

else

{

par->next = NULL;

tmp->next = NULL;

delete tmp;

return true;

}

return false;

}

string SymbolTable::find(string id)

{

int index = hashf(id);

Node \*start = head[index];

if (start == NULL)

return "-1";

while (start != NULL)

{

if (start->identifier == id)

{

start->print();

return start->scope;

}

start = start->next;

}

return "-1";

}

bool SymbolTable::insert(string id, string scope,

string Type, int lineno)

{

int index = hashf(id);

Node \*p = new Node(id, scope, Type, lineno);

if (head[index] == NULL)

{

head[index] = p;

cout << "\n"

<< id << " inserted";

return true;

}

else

{

Node \*start = head[index];

while (start->next != NULL)

start = start->next;

start->next = p;

cout << "\n"

<< id << " inserted";

return true;

}

return false;

}

int SymbolTable::hashf(string id)

{

int asciiSum = 0;

for (int i = 0; i < id.length(); i++)

{

asciiSum = asciiSum + id[i];

}

return (asciiSum % 100);

}

int main()

{

SymbolTable st;

string check;

cout << "\*\*\*\* SYMBOL\_TABLE \*\*\*\*\n";

if (st.insert("if", "local", "keyword", 4))

cout << " -successfully";

else

cout << "\nFailed to insert.\n";

if (st.insert("number", "global", "variable", 2))

cout << " -successfully\n\n";

else

cout << "\nFailed to insert\n";

check = st.find("if");

if (check != "-1")

cout << "Identifier Is present\n";

else

cout << "\nIdentifier Not Present\n";

if (st.deleteRecord("if"))

cout << "if Identifier is deleted\n";

else

cout << "\nFailed to delete\n";

if (st.modify("number", "global", "variable", 3))

cout << "\nNumber Identifier updated\n";

check = st.find("number");

if (check != "-1")

cout << "Identifier Is present\n";

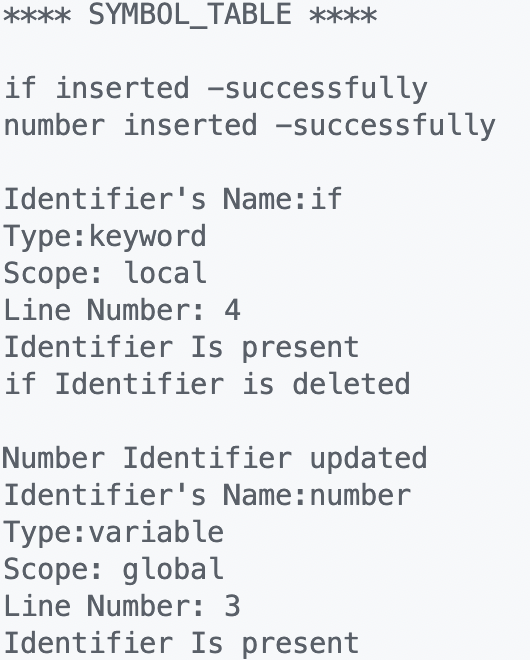
else

cout << "\nIdentifier Not Present";

return 0;

}

Output:



**Exercise 3**

**Question 1:**

Aim:

Write a C/C++ program to implement a recursive descent parser for the given grammar

Program:

#include <stdio.h>

#include <string.h>

#define SUCCESS 1

#define FAILED 0

// Function prototypes

int E(), Edash(), T(), Tdash(), F();

const char \*cursor;

char string[64];

int main()

{

puts("Enter the string");

scanf("%s", string); // Read input from the user

cursor = string;

puts("");

puts("Input Action");

puts("--------------------------------");

// Call the starting non-terminal E

if (E() && \*cursor == '\0')

{ // If parsing is successful and the cursor has reached the end

puts("--------------------------------");

puts("String is successfully parsed");

return 0;

}

else

{

puts("--------------------------------");

puts("Error in parsing String");

return 1;

}

}

// Grammar rule: E -> T E'

int E()

{

printf("%-16s E -> T E'\n", cursor);

if (T())

{ // Call non-terminal T

if (Edash()) // Call non-terminal E'

return SUCCESS;

else

return FAILED;

}

else

return FAILED;

}

// Grammar rule: E' -> + T E' | $

int Edash()

{

if (\*cursor == '+')

{

printf("%-16s E' -> + T E'\n", cursor);

cursor++;

if (T())

{ // Call non-terminal T

if (Edash()) // Call non-terminal E'

return SUCCESS;

else

return FAILED;

}

else

return FAILED;

}

else

{

printf("%-16s E' -> $\n", cursor);

return SUCCESS;

}

}

// Grammar rule: T -> F T'

int T()

{

printf("%-16s T -> F T'\n", cursor);

if (F())

{ // Call non-terminal F

if (Tdash()) // Call non-terminal T'

return SUCCESS;

else

return FAILED;

}

else

return FAILED;

}

// Grammar rule: T' -> \* F T' | $

int Tdash()

{

if (\*cursor == '\*')

{

printf("%-16s T' -> \* F T'\n", cursor);

cursor++;

if (F())

{ // Call non-terminal F

if (Tdash()) // Call non-terminal T'

return SUCCESS;

else

return FAILED;

}

else

return FAILED;

}

else

{

printf("%-16s T' -> $\n", cursor);

return SUCCESS;

}

}

// Grammar rule: F -> ( E ) | i

int F()

{

if (\*cursor == '(')

{

printf("%-16s F -> ( E )\n", cursor);

cursor++;

if (E())

{ // Call non-terminal E

if (\*cursor == ')')

{

cursor++;

return SUCCESS;

}

else

return FAILED;

}

else

return FAILED;

}

else if (\*cursor == 'i')

{

printf("%-16s F -> i\n", cursor);

cursor++;

return SUCCESS;

}

else

return FAILED;

}

Output:

