Logo

Description automatically generated

**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

**VISION**

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

**MISSION**

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

**Engineering Hardware – Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and encouraging inquisitive nature.

**Life – Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

**Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

**Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

**Digitization of Learning Processes**

The Institute provides seamless opportunities for innovative learning in all Engineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics, tutorials and related tools to create a platform for multi-disciplinary approach.

**Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities so that they become successful entrepreneurs and responsible citizens.

Logo

Description automatically generated

**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

**COMPUTER SCIENCE & ENGINEERING DEPARTMENT**

**VISION**

“To be centre of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values.”

**MISSION**

“To foster an open, multidisciplinary and highly collaborative research environment to produce world-class engineers capable of providing innovative solutions to real life problems and fulfil societal needs.”

**INDEX**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Experiment** | **Date** |  | **Marks** | | |  | **Signature** |
| **R1** | **R2** | **R3** | **R4** | **R5** |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Student Enrollment No: 02396402721 Student Name: Lakshay Sharma

Faculty Name: Dr. Deepak Gupta Signature:

**EXPERIMENT – 1**

**AIM:** 1.A. Implement the lexical analyzer generating tool and check whether the string is identifier/keyword/symbol by using rules of identifiers and keywords using lex tool.

**PROGRAM:**

%{

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

char\* keywords[] = {

"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"

};

bool isKeyword(char\* str) {

int numKeywords=sizeof(keywords)/sizeof(keywords[0]);

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

if (strcmp(str, keywords[i]) == 0) return true;

return false;

}

%}

%%

[a-zA-Z\_][a-zA-Z0-9\_]\* {

if (isKeyword(yytext)) {

printf("Keyword: %s\n", yytext);

} else {

printf("Identifier: %s\n", yytext);

}

}

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

[;{}(),=:] { printf("Symbol: %s\n", yytext); }

[ \t\n] { /\* Ignore whitespace and newline characters \*/ }

. { printf("Invalid: %s\n", yytext); }

%%

int yywrap() {}

int main() {

yylex();

return 0;

}

**OUTPUT:**

~/compilerdesignlab$ ~/compilerdesignlab$ lex main.c

~/compilerdesignlab$ gcc lex.yy.c

~/compilerdesignlab$ ./a.out

hi

Identifier: hi

if

Keyword: if

while

Keyword: while

10

Number: 10

^C

~/compilerdesignlab$

**AIM:** 1.B. Write a lex program to print “Compiler” when “hi” is input string.

**PROGRAM:**

%{

#include <stdio.h>

%}

%%

hi { printf("Compiler\n"); }

.\* { printf("Wrong"); }

%%

int yywrap(){}

int main() {

yylex();

return 0;

}

**OUTPUT:**

~/compilerdesignlab$ lex main.c

~/compilerdesignlab$ gcc lex.yy.c

~/compilerdesignlab$ ./a.out

hi

Compiler

Hello

Wrong

Hi

Wrong

hi

Compiler

^C

~/compilerdesignlab$

**AIM:** 1.C. Write a lex program to check whether number is even/odd.

**PROGRAM:**

%{

#include <stdio.h>

%}

%%

[0-9]+ {

int num = atoi(yytext);

if (num % 2 == 0) printf("%s is even\n", yytext);

else printf("%s is odd\n", yytext);

}

.|\n { /\* Ignore all other characters \*/ }

%%

int yywrap() {}

int main() {

yylex();

return 0;

}

**OUTPUT:**  
~/compilerdesignlab$ lex main.c

~/compilerdesignlab$ gcc lex.yy.c

~/compilerdesignlab$ ./a.out

23

23 is odd

45

45 is odd

42

42 is even

^C

~/compilerdesignlab$

**AIM:** 1.D. Write a lex program to print sum of 2 numbers.

**PROGRAM:**

%{

#include <stdio.h>

%}

%%

[0-9]+ {

int num1 = atoi(yytext);

int num2;

if (scanf("%d", &num2) == 1) {

int sum = num1 + num2;

printf("Sum: %d\n", sum);

}

}

.|\n { /\* Ignore all other characters \*/ }

%%

int yywrap() {}

int main() {

yylex();

return 0;

}

**OUTPUT:**  
~/compilerdesignlab$ lex main.c

~/compilerdesignlab$ gcc lex.yy.c

~/compilerdesignlab$ ./a.out

23

56

Sum: 79

32

41

Sum: 73

1

2

Sum: 3

^C

~/compilerdesignlab$

**VIVA-VOCE QUESTIONS:**

**Ques 1.** What is Lex?

**Ans 1.** Lex is a tool used to generate lexical analyzers (scanners) for parsing text input.

**Ques 2.** What is a Lex rule pattern?

**Ans 2.** A Lex rule pattern is a regular expression that describes the text to be matched in the input..

**Ques 3.** How do you specify the action to be taken when a Lex pattern is matched?

**Ans 3.** Actions in Lex are written in C code enclosed in curly braces ‘{}’ and placed after the pattern.

**Ques 4.** What does the double percentage symbols (%%) signify in a Lex program?

**Ans 4.** The double percentage symbols (%%) in a Lex program separate the lexical rules from the actions.

**Ques 5.** How can you compile and execute a Lex program?

**Ans 5.** Use lex to generate a C source file, then compile it using a C compiler (e.g., gcc) binary. and execute the resulting.

Top of Form

**EXPERIMENT – 2**

**AIM:** Write a program to check whether a given string belongs to a grammar or not.

**PROGRAMS:**

1. Grammar: S -> aS, S -> Sb, S -> ab

#include <bits/stdc++.h>

using namespace std;

int main() {

string str;

bool flag = true;

cout << "The grammar is: S->aS, S->Sb, S->ab" << endl;

cout << "Enter the string to be checked: ";

cin >> str;

int n = str.length();

if (str[0] == 'a' && str[n - 1] == 'b'){

for (int i = 1; i < str.length(); i++){

if (str[i] == 'b') flag = false;

else if (str[i] == 'a' && flag == false){

cout << "String is not accepted";

exit(0);

}

}

cout << "String is accepted";

}

else{

cout << "String is not accepted";

}

return 0;

}

**OUTPUT:**

The grammar is: S -> aS, S -> Sb, S -> ab

Enter the string to be checked: aaabbbb

String is accepted

Enter the string to be checked: a

String is not accepted

2. Grammar: S -> aSa, S -> bSb, S -> a, S -> b

#include <bits/stdc++.h>

using namespace std;

int main() {

string str;

bool flag = true;

cout << "The grammar is: S->aSa, S->bSb, S->a, S->b" << endl;

cout << "Enter the string to be checked: ";

cin >> str;

int n = str.length();

int a = 0, b = n - 1;

if (n % 2 != 0){

while (b > a){

if (str[a] == str[b]){

a++; b--;

}

else{

cout << "String is not accepted";

exit(0);

}

}

cout << "String is accepted";

} else{

cout << "String is not accepted";

}

return 0;

}

**OUTPUT:**

The grammar is: S->aSa, S->bSb, S->a, S->b

Enter the string to be checked: aaabbbb

String is not accepted

Enter the string to be checked: a

String is not accepted

3. Grammar: S -> aSbb, S -> abb

#include <iostream>

using namespace std;

int main() {

string str;

bool flag = true;

int a\_count = 0, b\_count = 0;

cout << "The grammar is: S->aSbb, S->abb" << endl;

cout << "Enter the string to be checked: ";

cin >> str;

int n = str.length();

if (str[0] == 'a' && str[n - 1] == 'b'){

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

if (str[i] == 'a' && flag == false){

cout << "String is not accepted";

exit(0);

} else if (str[i] == 'a' && flag == true) a\_count++;

else if (str[i] == 'b'){

b\_count++; flag = false;

}

}

if (b\_count == 2 \* a\_count) cout << "String is accepted";

else cout << "String is not accepted";

} else cout << "String is not accepted";

return 0;

}

**OUTPUT:**

The grammar is: S->aSbb, S->abb

Enter the string to be checked: aabbbb

String is accepted

Enter the string to be checked: aabbbab

String is not accepted

4. Grammar: S -> aSb, S -> ab

#include <iostream>

using namespace std;

int main() {

string str;

bool flag = true;

int a\_count = 0, b\_count = 0;

cout << "The grammar is: S->aSb, S->ab" << endl;

cout << "Enter the string to be checked: ";

cin >> str;

int n = str.length();

if (str[0] == 'a' && str[n - 1] == 'b'){

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

if (str[i] == 'a' && flag == false){

cout << "String is not accepted";

exit(0);

}

else if (str[i] == 'a' && flag == true) a\_count++;

else if (str[i] == 'b'){b\_count++; flag = false;}

}

if (b\_count == a\_count) cout << "String is accepted";

else cout << "String is not accepted";

}

else cout << "String is not accepted";

return 0;

}

**OUTPUT:**

The grammar is: S->aSb, S->ab

Enter the string to be checked: aaab

String is not accepted

Enter the string to be checked: aabb

String is accepted

**VIVA-VOCE QUESTIONS:**

**Ques 1.** What is the key feature of a CFG?

**Ans 1.** CFGs use production rules to generate strings in a language.

**Ques 2.** How do you determine if a string is in a CFG's language?

**Ans 2.** By constructing a parse tree for the string.

**Ques 3.** What are terminal symbols in a CFG?

**Ans 3.** Symbols that appear in the input string.

**Ques 4.** What is the significance of the Pumping Lemma for CFGs?

**Ans 4.** It helps identify non-context-free languages.

**Ques 5.** Are all programming languages context-free?

**Ans 5.** No, many programming languages have context-sensitive syntax.

**EXPERIMENT – 3**

**AIM:** Write a program to check whether a string includes keyword or not.

**PROGRAM:**

#include <iostream>

#include <string>

using namespace std;

int main() {

string input;

cout << "Enter a string: "; cin >> input;

string keywords[] = {

"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"

};

bool isKeyword = false;

for (const string& keyword : keywords) {

if (input == keyword) {

isKeyword = true;

break;

}

}

if (isKeyword) cout << input << " is a C++ keyword." << endl;

else cout << input << " is not a C++ keyword." << endl;

return 0;

}

**OUTPUT:**

Enter a string: do

do is a C++ keyword.

Enter a string: hi

hi is not a C++ keyword.

**EXPERIMENT – 4**

**AIM:** Write a program to remove left recursion from a grammar.

**PROGRAM:**

#include<iostream>

#include<string>

using namespace std;

int main() {

string ip,op1,op2,temp;

int sizes[10] = {};

char c;

int n,j,l;

cout<<"Enter the Parent Non-Terminal : ";

cin>>c;

ip.push\_back(c);

op1 += ip + "\'->";

ip += "->";

op2+=ip;

cout<<"Enter the number of productions : ";

cin>>n;

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

cout<<"Enter Production "<<i+1<<" : ";

cin>>temp;

sizes[i] = temp.size();

ip+=temp;

if(i!=n-1) ip += "|";

}

cout<<"Production Rule : "<<ip<<endl;

for(int i=0,k=3;i<n;i++) {

if(ip[0] == ip[k]) {

cout<<"Production "<<i+1<<" has left recursion."<<endl;

if(ip[k] != '#') {

for(l=k+1;l<k+sizes[i];l++) op1.push\_back(ip[l]);

k=l+1;

op1.push\_back(ip[0]);

op1 += "\'|";

}

} else {

cout<<"Production "<<i+1<<" does not have left recursion."<<endl;

if(ip[k] != '#') {

for(j=k;j<k+sizes[i];j++) op2.push\_back(ip[j]);

k=j+1;

op2.push\_back(ip[0]);

op2 += "\'|";

} else {

op2.push\_back(ip[0]);

op2 += "\'";

}

}

}

op1 += "#";

cout<<op2<<"\n"<<op1<<"\n";

return 0;

}

**OUTPUT:**

Enter the Parent Non-Terminal : 5

Enter the number of productions : 2

Enter Production 1 : Sa

Enter Production 2 : bS

Production Rule : 5->Sa|bS

Production 1 does not have left recursion.

Production 2 does not have left recursion.

S->SaS'|bSS'|

S'->#

**VIVA-VOCE QUESTIONS:**

**Ques 1.** What is left recursion in a context-free grammar (CFG)?

**Ans 1.** Left recursion in a CFG occurs when a non-terminal can directly or indirectly derive a production that starts with itself.

**Ques 2.** Why is left recursion problematic in a CFG?

**Ans 2.** Left recursion can lead to infinite loops in parsing, making it impossible to generate a valid parse tree for the grammar.

**Ques 3.** What is the general approach to remove left recursion from a CFG?

**Ans 3.** To remove left recursion, you create new non-terminals and rewrite productions so that the grammar no longer has any left-recursive productions.

**Ques 4.** How do you distinguish between left-recursive and right-recursive productions in a grammar?

**Ans 4.** A production is left-recursive if the first symbol on the right side is the same as the non-terminal on the left side. It is right-recursive if the non-terminal appears at the end.

**Ques 5.** What is the purpose of introducing new non-terminals when removing left recursion?

**Ans 5.** Introducing new non-terminals allows you to rewrite the grammar in a way that avoids left recursion while preserving the original language generated by the grammar.

**EXPERIMENT – 5**

**AIM:** Write a program to perform Left Factoring on a Grammar.

**PROGRAMS:**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

void leftFactoring(string nonTerminal, vector<string>& productions) {

// Step 1: Find the longest common prefix

string prefix;

for (int i = 0; i < productions[0].length(); i++) {

char ch = productions[0][i];

bool common = true;

for (int j = 1; j < productions.size(); j++)

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

common = false;

break;

}

if (common) prefix += ch;

else break;

}

// Step 2: Remove the common prefix from productions

for (int i = 0; i < productions.size(); i++)

productions[i] = productions[i].substr(prefix.length());

// Step 3: Generate new productions

cout << nonTerminal << " -> " << prefix << " " << nonTerminal << "'" << endl;

cout << nonTerminal << "' ->";

for (int i = 0; i < productions.size(); i++)

cout << " " << productions[i] << (i == productions.size() - 1 ? "" : " |");

cout << " | ε" << endl;

}

int main() {

string nonTerminal;

cout << "Enter the non-terminal symbol: ";

cin >> nonTerminal;

vector<string> productions;

cout << "Enter the productions (separated by space or comma): ";

string productionStr;

getline(cin >> ws, productionStr);

size\_t pos = 0;

string delimiter = " ";

while ((pos = productionStr.find(delimiter)) != string::npos) {

string production = productionStr.substr(0, pos);

productions.push\_back(production);

productionStr.erase(0, pos + delimiter.length());

}

if (!productionStr.empty()) productions.push\_back(productionStr);

leftFactoring(nonTerminal, productions);

return 0;

}

**OUTPUT:**

Enter the non-terminal symbol: S

Enter the productions (separated by space or comma): aS A bS aAa a b S

S -> S'

S' -> aA | bS | aAa | a | b | S | ε

**EXPERIMENT – 6**

**AIM:** Write a program to show all the operations of a stack.

**PROGRAMS:**

#include <iostream>

using namespace std;

class stack {

private:

int stk\_size,\*arr,index = -1;

public:

stack(int size) {stk\_size = size; arr = new int[size];}

void push(int elem) {

if (++index < stk\_size)

arr[index] = elem;

else {

cout << "Stack is full\n";

index--;

}

}

void pop() {

if (index >= 0)

cout << "pop:" << arr[index--] << "\n";

else cout << "Stack is empty\n";

}

void top() {cout << "Top:" << arr[index] << "\n";}

void size() {cout << "Size:" << index+1 << "\n";}

void isempty() {cout << "Is empty:" << (index < 0 ? "true" : "false") << "\n";}

};

int main() {

int size;

cout << "Enter the stack size:"; cin >> size;

stack obj = stack(size);

cout << "Adding elements {10,14,19} to stack\n";

obj.push(10);

obj.push(14);

obj.push(19);

obj.top();

obj.pop();

obj.pop();

obj.pop();

obj.pop();

obj.isempty();

return 0;

}

**OUTPUT:**

Enter the stack size:3

Adding elements {10,14,19} to stack

Top:19

pop:19

pop:14

pop:10

Stack is empty

Is empty:true