Sri Indu
College of Engineering & Technology
UGC Autonomous Institution
Recognized under 2(f) & 12(B) of UGC Act 1956,
NAAC, Approved by AICTE &
Permanently Affiliated to JNTUH

Estd.2004

NAAC
NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL

COMPILER DESIGN LAB MANUAL

IV Year– I Semester

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR 2024-25

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY
B. TECH – COMPUTER SCIENCE AND ENGINEERING

INSTITUTION VISION

To be a premier Institution in Engineering & Technology and Management with competency, values and socialconsciousness.

INSTITUTION MISSION

IM1 Provide high quality academic programs, training activities and research facilities.

IM2 Promote Continuous Industry-Institute interaction for employability, Entrepreneurship, leadership andresearch aptitude among stakeholders.

IM3 Contribute to the economical and technological development of the region, state and nation.

DEPARTMENT VISION

To be a recognized knowledge center in the field of Information Technology with self - motivated, employable engineers to society.

DEPARTMENT MISSION

The Department has following Missions:

DM1 To offer high quality student centric education in Information Technology.

DM2 To provide a conducive environment towards innovation and skills.

DM3 To involve in activities that provide social and professional solutions.

DM4 To impart training on emerging technologies namely cloud computing and IOT with involvementof stake holders.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: Higher Studies: Graduates with an ability to apply knowledge of Basic sciences andprogramming skills in their career and higher education.

PEO 2: Lifelong Learning: Graduates with an ability to adopt new technologies for ever changingIT industry needs through Self-Study, Critical thinking and Problem solving skills.

PEO 3: Professional skills: Graduates will be ready to work in projects related to complex problems involving multi-disciplinary projects with effective analytical skills.

PEO 4: Engineering Citizenship: Graduates with an ability to communicate well and exhibitsocial, technical and ethical responsibility in process or product.

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY
B. TECH – COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES (POs) & PROGRAM SPECIFIC OUTCOMES

PO	Description
PO 1	**Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO 2	**Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO 3	**Design / development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO 4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO 5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6	**The engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO 9	**Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO 11	**Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological Change

Program Specific Outcomes(PSOs)	
PSO 1	**Software Development:** To apply the knowledge of Software Engineering, Data Communication, Web Technology and Operating Systems for building IOT and CloudComputing applications.
PSO 2	**Industrial Skills Ability:** Design, develop and test software systems for world-wide network of computers to provide solutions to real world problems.
PSO 3	**Project Implementation:** Analyze and recommend the appropriate IT infrastructure requiredfor the implementation of a project.

Course Outcomes (COS)	**Program Outcomes (POs)**												**Program Specific Outcomes (PSOs)**		
	PO1	PO2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	P10	P1 1	P12	PSO1	PSO2	PS O 3
C32L1.1	1	1	2	-	3	-	-	-	-	-	-	-	-	-	-
C32L1.2	1	-	1	-	3	-	-	-	-	-	-	-	-	1	-
C32L1.3	1	1	2	1	3	-	-	-	-	-	-	-	-	1	-
C32L1.4	1	1	1	2	3	-	-	-	-	-	-	-	-	1	1
C32L1.5	1	-	1	1	3	-	-	-	-	-	-	-	-	-	-
C32L1	1	1	1.4	2	3	-	-	-	-	-	-	-	1	1	1

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), thosewho come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure,Program, Expected Output, etc.,) filled in for the lab session.
   b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed inthe lab.
   c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allottedto you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book,and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should beutilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severepunishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody foundloitering outside the lab / class without permission during working hours will be treated seriously andpunished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, New Delhi)

B.Tech. – IV Year – ISemester

L	T	P	C
0	0	2	1

(R22CSE4127) COMPILER DESIGN LAB

Course Objectives:

- To provide hands-on experience on web technologies
- To develop client-server application using web technologies
- To introduce server-side programming with Java servlets and JSP
- To understand the various phases in the design of a compiler.
- To understand the design of top-down and bottom-up parsers.
- To understand syntax directed translation schemes.
- To introduce lex and yacc tools.

Course Outcomes:

- Design and develop interactive and dynamic web applications using HTML, CSS, JavaScriptand XML
- Apply client-server principles to develop scalable and enterprise web applications.
- Ability to design, develop, and implement a compiler for any language.
- Able to use lex and yacc tools for developing a scanner and a parser.
- Able to design and implement LL and LR parsers.

List of Experiments

Compiler Design Experiments

1. Write a LEX Program to scan reserved word & Identifiers of C Language
2. Implement Predictive Parsing algorithm
3. Write a C program to generate three address code.
4. Implement SLR(1) Parsing algorithm
5. Design LALR bottom up parser for the given language

<program> ::= <block>
<block> ::= { <variabledefinition> <slist> }
| { <slist> }
<variabledefinition> ::= int <vardeflist> ;
<vardeflist> ::= <vardec> | <vardec> , <vardeflist>
<vardec> ::= <identifier> | <identifier> [<constant>]
<slist> ::= <statement> | <statement> ; <slist>
<statement> ::= <assignment> | <ifstatement> | <whilestatement>
| <block> | <printstatement> | <empty>
<assignment> ::= <identifier> = <expression>
| <identifier> [<expression>] = <expression>
<ifstatement> ::= if <bexpression> then <slist> else <slist> endif
| if <bexpression> then <slist> endif
<whilestatement> ::= while <bexpression> do <slist> enddo
<printstatement> ::= print (<expression>)
<expression> ::= <expression> <addingop> <term> | <term> | <addingop> <term>
<bexpression> ::= <expression> <relop> <expression>

<relop> ::= < | <= | == | >= | > | !=
<addingop> ::= + | -
<term> ::= <term> <multop> <factor> | <factor>
<multop> ::= * | /
<factor> ::= <constant> | <identifier> | <identifier> [<expression>]
| (<expression>)
<constant> ::= <digit> | <digit> <constant>
<identifier> ::= <identifier> <letterordigit> | <letter>
<letterordigit> ::= <letter> | <digit>
<letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit> ::= 0|1|2|3|4|5|6|7|8|9
<empty> has the obvious meaning
Comments (zero or more characters enclosed between the standard C/Java-style comment brackets /*...*/) can be inserted. The language has rudimentary support for 1-dimensional arrays. The declaration int a[3] declares an array of three elements, referenced as a[0], a[1] and a[2]. Notealso that you should worry about the scoping of names.
A simple program written in this language is:

```
{ int a[3],t1,t2; t1=2;
a[0]=1; a[1]=2; a[t1]=3;
t2=-(a[2]+t1*6)/(a[2]-t1);
if t2>5 then print(t2); else
{
int t3; t3=99; t2=-25;
print(-t1+t2*t3); /* this is a comment
on 2 lines */
}
endif
}
```

List of Experiments

1. Write a program to simulate a machine known as the Deterministic Finite Automata(DFA).
2. write a program for dividing the given input program into lexems.
3. Write a C program to check the string of a given grammar.
4. C Program To Remove the Left Recursion from a Given Grammer | Elimination of Left Recursion
5. Write a program to implement predictive parser
6. Write a C program to show the implementation of shift-Reduce Parser.
7. Write a LEX Program to scan reserved word & Identifiers of C Language
8. Implement Predictive Parsing algorithm
9. Write a C program to generate three address code.
10. Implement SLR(1) Parsing algorithm
11. Design LALR bottom up parser for the given language

Reference Books:

1. Introduction to automata theory,languages and computation,3rd edition,John E Hoperoft,Rajeev,Jeffrey , D Ullman,Pearson education
2. Compilers principles and techniques nd tools-Alfred V,Aho,Monica

COMPILER DESIGN LAB PROGRAMS

1. Write a program to simulate a machine known as the Deterministic Finite Automata(DFA).
Aim:program to simulate a machine known as the Deterministic Finite Automata(DFA).
Source code:

```
#include <stdio.h> #include <stdlib.h> struct node{
int id_num; int st_val;
struct node *link0; struct node *link1;
};
struct node *start, *q, *ptr; int vst_arr[100], a[10];
int main(){
int count, i, posi, j; char n[10];
printf("=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=\n");
printf("Enter the number of states in the m/c:"); scanf("%d",&count);
q=(struct node *)malloc(sizeof(struct node)*count); for(i=0;i<count;i++){
(q+i)->id_num=i;
printf("State Machine::%d\n",i);
printf("Next State if i/p is 0:"); scanf("%d",&posi);
(q+i)->link0=(q+posi); printf("Next State if i/p is 1:"); scanf("%d",&posi);
(q+i)->link1=(q+posi);
printf("Is the state final state(0/1)?"); scanf("%d",&(q+i)->st_val);
}
printf("Enter the Initial State of the m/c:"); scanf("%d",&posi);
start=q+posi;
printf("=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=\n");
while(1){
printf("=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=\n");
printf("Perform String Check(0/1):"); scanf("%d",&j);
if(j){ ptr=start;
printf("Enter the string of inputs:"); scanf("%s",n);
posi=0; while(n[posi]!='\0'){
a[posi]=(n[posi]-'0');
//printf("%c\n",n[posi]);
//printf("%d",a[posi]); posi++;
} i=0;
printf("The visited States of the m/c are:");
do{
vst_arr[i]=ptr->id_num; if(a[i]==0){
ptr=ptr->link0;
}
else if(a[i]==1){ ptr=ptr->link1;
}
else{
printf("iNCORRECT iNPUT\n"); return;
}
printf("[%d]",vst_arr[i]); i++;
}while(i<posi); printf("\n");
printf("Present State:%d\n",ptr->id_num); printf("String Status:: ");
if(ptr->st_val==1) printf("String Accepted\n"); else
printf("String Not Accepted\n");
}
else return 0;
}
printf("=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=\n");
return 0;
}
```

Output:

Enter the number of states in the m/c:2
State machine:0
Next state if input is 0:1 Next state if input is 1:0
Is the state final state(0/1)?1 State machine:1
Next state if input is 0:0 Next state if input is 1:1
Is the state final state(0/1)?0 Enter the initial state of the m/c:0

Perform string check(0/1):1 Enter the string of inputs:1
The visited states of the m/c are:(0) Present state:0
String status: string Accepted

Viva Questions:

What is finite automata
What are the different types of finite automata
Define DFA
How many tuples does DFA contains
What are the applications of finite automata

2. write a program for dividing the given input program into lexems.

Aim:program for dividing the given input program into lexems.

Source code:

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

// Returns 'true' if the character is a DELIMITER. bool isDelimiter(char ch)
{

if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == ',' || ch == ';' || ch == '>' || ch == '<' || ch == '='
|| ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}') return (true);
return (false);
}
// Returns 'true' if the character is an OPERATOR. bool isOperator(char ch)
{
if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '>' || ch == '<' || ch == '=')
return (true); return (false);
}
// Returns 'true' if the string is a VALID IDENTIFIER. bool validIdentifier(char* str)
{
if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
str[0] == '3' || str[0] == '4' || str[0] == '5' ||
str[0] == '6' || str[0] == '7' || str[0] == '8' || str[0] == '9' || isDelimiter(str[0]) == true) return (false);
return (true);
}
// Returns 'true' if the string is a KEYWORD. bool isKeyword(char* str)
{
if (!strcmp(str, "if") || !strcmp(str, "else") ||
!strcmp(str, "while") || !strcmp(str, "do") ||
!strcmp(str, "break") ||
!strcmp(str, "continue") || !strcmp(str, "int")
|| !strcmp(str, "double") || !strcmp(str, "float")
|| !strcmp(str, "return") || !strcmp(str, "char")
|| !strcmp(str, "case") || !strcmp(str, "char")
                || !strcmp(str, "sizeof") || !strcmp(str, "long")
                || !strcmp(str, "short") || !strcmp(str, "typedef")

                || !strcmp(str, "switch") || !strcmp(str, "unsigned")

                || !strcmp(str, "void") || !strcmp(str, "static")

                || !strcmp(str, "struct") || !strcmp(str, "goto"))


return (true); return (false);
}
// Returns 'true' if the string is an INTEGER. bool isInteger(char* str)
{
int i, len = strlen(str);
if (len == 0)
return (false);
for (i = 0; i < len; i++)
if (str[i] != '0' && str[i] != '1' && str[i] != '2'
&& str[i] != '3' && str[i] != '4' && str[i] != '5'
&& str[i] != '6' && str[i] != '7' && str[i] != '8'
&& str[i] != '9' || (str[i] == '-' && i > 0)) return (false);
}
```

```
return (true);
}
// Returns 'true' if the string is a REAL NUMBER. bool isRealNumber(char* str)
{
int i, len = strlen(str); bool hasDecimal = false;

if (len == 0)
return (false);
for (i = 0; i < len; i++) {
if (str[i] != '0' && str[i] != '1' && str[i] != '2'
&& str[i] != '3' && str[i] != '4' && str[i] != '5'
&& str[i] != '6' && str[i] != '7' && str[i] != '8'
&& str[i] != '9' && str[i] != '.' ||
(str[i] == '-' && i > 0)) return (false);
if (str[i] == '.') hasDecimal = true;
}
return (hasDecimal);
}
// Extracts the SUBSTRING
char* subString(char* str, int left, int right)
{
int i;
char* subStr = (char*)malloc(
sizeof(char) * (right - left + 2));
for (i = left; i <= right; i++) subStr[i - left] = str[i];
subStr[right - left + 1] = '\0'; return (subStr);
}
// Parsing the input STRING. void parse(char* str)
{
int left = 0, right = 0; int len = strlen(str);
while (right <= len && left <= right) {
if (isDelimiter(str[right]) == false) right++;
if (isDelimiter(str[right]) == true && left == right) { if (isOperator(str[right]) == true)
printf("'%c' IS AN OPERATOR\n", str[right]);
right++;
left = right;
} else if (isDelimiter(str[right]) == true && left != right
|| (right == len && left != right)) { char* subStr = subString(str, left, right - 1);
if (isKeyword(subStr) == true) printf("'%s' IS A KEYWORD\n", subStr);
else if (isInteger(subStr) == true) printf("'%s' IS AN INTEGER\n", subStr);
else if (isRealNumber(subStr) == true) printf("'%s' IS A REAL NUMBER\n", subStr);
else if (validIdentifier(subStr) == true
&& isDelimiter(str[right - 1]) == false) printf("'%s' IS A VALID IDENTIFIER\n", subStr);
else if (validIdentifier(subStr) == false
&& isDelimiter(str[right - 1]) == false) printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);
left = right;
}
}
return;
}
// DRIVER FUNCTION
int main()
{
// maximum length of string is 100 here char str[100] = "int a = b + 1c; ";
parse(str); // calling the parse function
return (0);}
```

Output:

'int' IS A KEYWORD

'a' IS A VALID IDENTIFIER

'=' IS AN OPERATOR

'b' IS A VALID IDENTIFIER

'+' IS AN OPERATOR

'1c' IS NOT A VALID IDENTIFIER

Viva questions:

1. What is lexeme
2. What is Token
3. What is pattern
4. Give any exmples for token
5. Define lexical analyzer

3. Write a C program to check the string of a given grammar.

Aim: program to check the string of a given grammar.

Source code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main() {
char string[50]; int flag,count=o; clrscr();
printf("The grammar is: S->aS, S->Sb, S->ab\n"); printf("Enter the string to be checked:\n"); gets(string);
if(string[0]=='a') {
flag=0;
for (count=1;string[count-1]!='\0';count++) {
if(string[count]=='b') { flag=1; continue;
} else if((flag==1)&&(string[count]=='a')) {
printf("The string does not belong to the specified grammar"); break;
} else if(string[count]=='a')
continue; else if(flag==1)&&(string[count]='\0')) { printf("String accepted…..!!!!");
break;
} else {
printf("String not accepted");
}
}
}
getch();
}
```

Output:

```
The grammer is:
S->aS S->Sb
S->ab
Enter the string to be checked:
aab
String accepted….!!!!
```

Viva Questions:

1. Define Regular grammar
2. What is context free grammar
3. Define tuples in CFG
4. Give example for production rules
5. Define ambiguous grammar

4. Write a Program To Remove the Left Recursion from a Given Grammar | Elimination of Left Recursion

Aim: Program To Remove the Left Recursion from a Given Grammer | Elimination of Left Recursion

Source code:

```
#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<<endl; cout<<op1<<endl; return 0;}
```

Output:

Enter the Parent Non-Terminal: E Enter the number of productions : 3 Enter Production 1: E+T
Enter Production 2: T Enter Production 3: #
Production Rule : E->E+T|T|#
Production 1 has left recursion. Production 2 does not have left recursion. Production 3 does not have left recursion. E->TE'|E'
E'->+TE'|

Viva Questions:

1. What is left recursion
2. Define left factoring
3. Why we eliminate left recursion
4. Define top down parser
5. Explain elements of LL(1) parser

5. Write a C program on recursive descent parser

Aim:program on recursive descent parser

Source code:

```
 #include<stdio.h>
#include<conio.h> #include<string.h> char input[100]; int i,l;
void main()
{
clrscr();
printf("\nRecursive descent parsing for the following grammar\n");
 printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->*FT'/@\nF->(E)/ID\n"); printf("\nEnter the string to be
checked:");
gets(input); if(E())
{
if(input[i+1]=='\0') printf("\nString is accepted");
}
else
printf("\nString is not accepted"); getch();


} E()
{
if(T())
{
if(EP())
return(1);
else return(0);
} EP()
{
if(input[i]=='+')
{
i++; if(T())
{ if(EP())
return(1); else return(0);
}
Else Return(0);
}
Else Return(1)
}
T()
{ if(F())
{
return(0);
}
else return(1); if(TP())
return(1); else return(0);
} TP()
{
if(input[i]=='*')
{
i++; if(F())
{ if(TP())
return(1); else return(0);
}
else
return(0);
}
```

```
else return(1);
}
F()
{
if(input[i]=='(')
{
i++; if(E())
{
if(input[i]==')')
{ i++;
return(1);
}
Else return(0);
}
else if(input[i]>='a'&&input[i]<='z'||input[i]>='A'&&input[i]<='Z')
{ i++;
return(1);
}
Else return(0);
}
```

Output:

```
Recursive descent parsing for the following grammar E->TE'
E'->+TE'/@ T->FT'
T'->*FT'/@ F->(E)/ID
Enter the string to be checked:(a+b)*c String is accepted
Recursive descent parsing for the following grammar E->TE'
E'->+TE'/@ T->FT'
T'->*FT'/@ F->(E)/ID
Enter the string to be checked:a/c+d String is not accepted
```

Viva Questions:

1. Define recursive descent parser
2. Explain FIRST rules
3. Define FOLLOW
4. What is parsing table
5. What are the steps involved in Recursive descent parser construction

6. Write a C program to show the implementation of shift-Reduce Parser.

Aim: program to show the implementation of shift-Reduce Parser.

Source code:

```
#include<stdio.h>
#include<stdlib.h> #include<conio.h> #include<string.h>
char ip_sym[15],stack[15]; int ip_ptr=0,st_ptr=0,len,i; char temp[2],temp2[2]; char act[15];
void check(); void main()
{
clrscr();
printf("\n\t\t SHIFT REDUCE PARSER\n"); printf("\n GRAMMER\n");
printf("\n E->E+E\n E->E/E"); printf("\n E->E*E\n E->a/b"); printf("\n enter the input symbol:\t");
gets(ip_sym);
printf("\n\t stack implementation table"); printf("\n stack \t\t input symbol\t\t action"); printf("\n   \t\t
          \t\t       \n"); printf("\n $\t\t%s$\t\t\t--",ip_sym); strcpy(act,"shift");
temp[0]=ip_sym[ip_ptr]; temp[1]='\0'; strcat(act,temp); len=strlen(ip_sym)for(i=0;i<=len-1;i++)
{
stack[st_ptr]=ip_sym[ip_ptr]; stack[st_ptr+1]='\0'; ip_sym[ip_ptr]=' ';
ip_ptr++;
printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act); strcpy(act,"shift");
temp[0]=ip_sym[ip_ptr]; temp[1]='\0'; strcat(act,temp); check();
st_ptr++;
 }
st_ptr++; check();
}void check()
{
int flag=0; temp2[0]=stack[st_ptr]; emp2[1]='\0';
if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))
{

stack[st_ptr]='E';
if(!strcmpi(temp2,"a"))

printf("\n $%s\t\t%s$\t\t\tE->a",stack,ip_sym); else
printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip_sym); flag=1;
}

if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"*"))||(!strcmpi(temp2,"/")))

{

flag=1;

}
 if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E*E")))
 {
strcpy(stack,"E"); st_ptr=0; if(!strcmpi(stack,"E+E"))
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym); else
if(!strcmpi(stack,"E\E"))

printf("\n $%s\t\t%s$\t\t\tE->E\E",stack,ip_sym); else
 if(!strcmpi(stack,"E*E"))

 printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym); else

printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym); flag=1;
}
if(!strcmpi(stack,"E")&&ip_ptr==len)
```

```
{

printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym); getch();
exit(0);

}

if(flag==0)

{

printf("\n%s\t\t\t%s\t\t reject",stack,ip_sym); exit(0);
}
return;
}
```

Output:

```
SHIFT REDUCE PARSER GRAMMER
E->E+E E->E/E E->E*E
E->a/b
Enter the input symbol: a+b

Stack  Implementation  Table

Stack   Input Symbol  Action
--------    --------------------    ----------
      $         a+b$              --

      $a          +b$             shift a
      $E          +b$             E->a

      $E+         b$              shift +

      $E+b        $               shift b

      $E+E        $               E->b

      $E          $               E->E+E

      $E          $               ACCEPT
```

Viva Questions:

1. Define bottom up parser
2. How many types of LR parsers are there
3. Define Action
4. Define Goto
5. Explain the steps involved in LR parser construction

7. Write a LEX Program to scan reserved word & Identifiers of C Language

Aim: Write a LEX program to scan reserved word and identifier of C language.

Source Code:

```
#include  <stdio.h>
#include<conio.h>
#include  <ctype.h>
#include <string.h>
int i;
int isKeyword(char str[]) {
const char *keywords[] = {
"int", "char", "void", "return", "if",
"else", "for", "while", "do", "break", "continue",
"switch", "case", "default", "goto", "long", "float",
"double", "short", "struct", "union", "enum", "typedef",
"static", "extern", "sizeof"
};
int total_keywords = 26;
for(i=0;i<total_keywords;i++)
{
if(strcmp(str, keywords[i]) == 0)
{
return 1;
}
}
return 0;
}
int isValidIdentifier(char str[])
{
if (isKeyword(str)) {
return 0;
}
if (!(isalpha(str[0]) || str[0] == '_')) {
return 0;
}
for(i=1;i<strlen(str);i++)
{
if (!(isalnum(str[i]) || str[i] == '_')) {
return 0;
}
}
return 1;
}
int main() {
char str[100];
printf("Enter a string keyword: ");
scanf("%s", str);
if (isKeyword(str)) {
printf("\"%s\" is a valid keyword.\n", str);
} else {
printf("\"%s\" is not a valid keyword.\n", str);
}
{
char str[100];
printf("Enter a string identifier: ");
scanf("%s", str);
if (isValidIdentifier(str)) {
```

```
printf("\"%s\" is a valid identifier.\n", str);
} else {
printf("\"%s\" is not a valid identifier.\n", str);
}
return 0;
}
}
```

Output:

Enter a string keyword: int
"int" is a valid keyword
Enter a string identifier: ab
"ab" is a valid identifier.

Viva Questions:

1. What is keyword
2. What is identifier
3. Define syntax analysis
4. What is the role of compiler
5. Define input buffering

8. Implement Predictive Parsing algorithm

Aim: Implement Predictive Parsing algorithm

Source code;

```
#include<stdio.h> #include<ctype.h> #include<string.h> #include<stdlib.h> #define SIZE 128
#define NONE -1 #define EOS '\0' #define NUM 257
#define KEYWORD 258
#define ID 259
#define DONE 260
#define MAX 999 charlexemes[MAX]; charbuffer[SIZE]; intlastchar=-1; intlastentry=0;
inttokenval=DONE; intlineno=1; intlookahead; structentry
{
char*lexptr; inttoken;
}
symtable[100]; structentry
keywords[]=
{"if",KEYWORD,"else",KEYWORD,"for",KEYWORD,"int",KEYWORD,"float",KEYWORD,
"double",KEYWORD,"char",KEYWORD,"struct",KEYWORD,"ret
urn",KEYWORD,0,0
};
voidError_Message(char*m)
{
fprintf(stderr,"line %d, %s \n",lineno,m); exit(1);
}
intlook_up(chars[ ])
{
intk;
for(k=lastentry; k>0; k--) if(strcmp(symtable[k].lexptr,s)==0)
returnk;
return0;
}
intinsert(chars[ ],inttok)
{
intlen; len=strlen(s); if(lastentry+1>=MAX)
Error_Message("Symbpl table is full"); if(lastchar+len+1>=MAX)
Error_Message("Lexemes array is full"); lastentry=lastentry+1; symtable[lastentry].token=tok;
symtable[lastentry].lexptr=&lexemes[lastchar+1]; lastchar=lastchar+len+1;
strcpy(symtable[lastentry].lexptr,s); returnlastentry;
}
/*void Initialize()
{
}*/struct entry *ptr; for(ptr=keywords;ptr->token;ptr+1)
insert(ptr->lexptr,ptr->token);
intlexer()
{
intt; intval,i=0; while(1)
{
t=getchar();
if(t==' '||t=='\t'); elseif(t=='\n')
lineno=lineno+1; elseif(isdigit(t))
{
ungetc(t,stdin); scanf("%d",&tokenval); returnNUM;
}
elseif(isalpha(t))
{
while(isalnum(t))
{
```

```
buffer[i]=t; t=getchar(); i=i+1; if(i>=SIZE)
Error_Message("Compiler error");
}
buffer[i]=EOS; if(t!=EOF)
ungetc(t,stdin); val=look_up(buffer); if(val==0)
val=insert(buffer,ID); tokenval=val; returnsymtable[val].token;
}
elseif(t==EOF) returnDONE;
}
}
tokenval=NONE; returnt;
voidMatch(intt)
{
if(lookahead==t) lookahead=lexer();
else
Error_Message("Syntax error");
}
voiddisplay(intt,inttval)
{
if(t=='+'||t=='-'||t=='*'||t=='/') printf("\nArithmetic Operator: %c",t);
elseif(t==NUM)
printf("\n Number: %d",tval); elseif(t==ID)
printf("\n Identifier: %s",symtable[tval].lexptr);
else
printf("\n Token %d tokenval %d",t,tokenval);
}
voidF()
{
//void E(); switch(lookahead)
{
case'(':
Match('('); E();
Match(')'); break;
caseNUM :
display(NUM,tokenval); Match(NUM);
break; caseID :
display(ID,tokenval); Match(ID);
break; default:
Error_Message("Syntax error");
}
}
voidT()
{
intt; F();
while(1)
{
switch(lookahead)
{
case'*':
t=lookahead; Match(lookahead); F();
display(t,NONE); continue;
case'/':
t=lookahead; Match(lookahead); display(t,NONE); continue;
default:
return;
}
}
```

```
}
voidE()
{
intt; T();
while(1)
{
switch(lookahead)
{
case'+':
t=lookahead;
Match(lookahead); T();
display(t,NONE); continue;
case'-':
t=lookahead; Match(lookahead); T();
display(t,NONE); continue;
default:
return;
}
}
}
voidparser()
{
lookahead=lexer(); while(lookahead!=DONE)
{
E();
Match(';');
}
}
intmain()
{
charans[10];
printf("\n Program for recursive descent parsing "); printf("\n Enter the expression ");
printf("And place ; at the end\n"); printf("Press Ctrl-Z to terminate\n"); parser();<br>return0;
}
```

Output-

```
Program for recursive descent parsing
 Enter the expression And place ; at the end
Press Ctrl-Z to terminate
a*b+c;

 Identifier: a
 Identifier: b
Arithmetic Operator: *
 Identifier: c
Arithmetic Operator: +
5*7;

 Number: 5
 Number: 7
Arithmetic Operator: *
*2;
line 5, Syntax error
```

Viva Questions:

1. What is predictive parser
2. Define top down parser
3. What is parse tree
4. How many derivations we have in a grammar
5. What is look ahead

9. Write a C program to generate three address code.

Aim:-To generate three address codes.

Source Code:

```
#include<stdio.h>
#include<string.h>
void pm();
void plus();
void div();
int i,ch,j,l,addr=100;
char ex[10], exp[10] ,exp1[10],exp2[10],id1[5],op[5],id2[5];
void main()
{
clrscr();
while(1)
{
printf("\n1.arithmetic\n2.Exit\nEnter the choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the expression with arithmetic operator:");
scanf("%s",ex);
strcpy(exp,ex);
l=strlen(exp);
exp1[0]='\0';

for(i=0;i<l;i++)
{
if(exp[i]=='+'||exp[i]=='-')
{
if(exp[i+2]=='/'||exp[i+2]=='*')
{
pm();
break;
}
else
{
plus();
break;
}
}
else if(exp[i]=='/'||exp[i]=='*')
{
div();
break;
}
}
break;
case  2:
exit(0);
}
}
}
void pm()
{
strrev(exp);
j=l-i-1;
```

```
strncat(exp1,exp,j);
strrev(exp1);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);
}
void div()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
void plus()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
```

Output:

```
1.arithmetic
2.Exit
Enter the choice:1
Enter the expression with arithmetic operator: a*b-c
Three address code:
temp=a*b
temp1=temp-c
1.arithmetic
2.Exit
Enter the choice:1
Enter the expression with arithmetic operator: a/b*c
Three address code:
temp=a/b
temp1=temp*c
```

Viva Questions:

1. define intermediate code generator
2. What is three address code
3. What is quadraple give example
4. Define code optimiztion
5. Explain about machine independent lnguage

10. Implement SLR(1) Parsing algorithm.

Aim:- To implement SLR() parsing algorithm. ALGORITHM / PROCEDURE :

Source Code :

```
#include<stdio.h>
char tab[12][9][2]={"S5","N","N","S4","N","N","1","2","3",
"N","S6","N","N","N","Z","0","0","0",
"N","R2","S7","N","R2","R2","0","0","0",
"N","R4","R4","N","R4","R4","0","0","0",
"S5","N","N","S4","N","N","8","2","3",
"N","R6","R6","N","R6","R6","0","0","0",
"S5","N","N","S4","N","N","0","9","3",
"S5","N","N","N","S4","N","0","0","X",
"N","S6","N","N","SY","N","0","0","0",
"N","R1","S7","N","R1","R1","0","0","0",
"N","R3","R3","N","R3","R3","0","0","0",
"N","R5","R5","N","R5","R5","0","0","0"};

/*Z=accept; N->error; X->10;
Y->11;*/
char prod[7][4]={"0","E+T","T","T*F","F","(E)","d"};
char index[12]={'0','1','2','3','4','5','6','7','8','9','X','Y'};
char term[9]={'d','+','*','(',')','$','E','T','F'};
introw,col,st_pt=0,ip_pt=0; char input[10], stack[20]; clrscr();
void main()
{
intj,k;
printf("\n enter input string :"); scanf("%s", input);
strcat(input,"$");
stack[0]='0'; for(j=0;j<7;j++) strcat(prod[j],"\0");
printf("\n STACK       INPUT \n \n"); while(1)
{
for(k=0;k<=st_pt; k++) printf("%c", stack[k]); printf("         ");
for(k=ip_pt;input[k-1]!='$';k++)
printf("%c", input[k]); printf("\n"); row=is_index(stack[st_pt]); col=is_term(input[ip_pt]);
if(tab[row][col][0]=='S')
shift(tab[row][col][1]); else if(tab[row][col][0]=='R')
reduce(tab[row][col][1]);
else if(tab[row][col][0]=='Z')
{
printf (" \n success"); getch();
exit(0);
}
else if(tab[row][col][0]=='N')
{
printf("\n error"); getch();
exit(0);
}
}
}
shift(char ch)
{
st_pt++; stack[st_pt++]=input[ip_pt++]; stack[st_pt]=ch;
}
reduce(char ch)
{
intk,prno,prlen,rowno,colno; for(k=1;k<7;k++) if(index[k]==ch)
prno=k; prlen=strlen(prod[prno]); for(k=1;k<=2*prlen;k++) st_pt--;
```

```
st_pt++; if(prno==1||prno==2) stack[st_pt]='E';
else if(prno==3||prno==4) stack[st_pt]='T';
else if(prno==5||prno==6) stack[st_pt]='F';
rowno=is_index(stack[st_pt-1]); colno=is_term(stack[st_pt]); stack[++st_pt]=tab[rowno][colno][0];
}
is_index(char ch)
{
int k;
for (k=0;k<=9;k++) if(index[k]==ch) return(k); if(ch=='X') return(10);
else
{
    if(ch=='Y')
    return(11);
    }
    is_term(char ch)
    {
    int k;
    for(k=0;k<9;k++)
    if(term[k]==ch)
    return(k);
    }
```

Output:

```
enter input
string(d*d) Stack                    input
                        (d+d)$
O(4                          d+d)$
0(4d5                        +d)$
0(4f3                        +d)$
0(4T2                        +d)$
0(4E8                        +d)$ d)
0(4E8+6                      $
0(4E8+6d5                    )$
0(4E8+6F3                    )$
0(4E8+6T9                    )$
0(4e8)Y                      $
0f3 0T2                      $           $
0E1                                      $
success
```

Viva Question:

1. Define SLR(1)
2. Define bottom up parser
3. What is operator precedense parser
4. Define Action
5. What is augmented grammer

11. LALR bottom up parser for the given language

Aim: To Design and implement an LALR bottom up Parser for checking the syntax of the Statements in the given language.

Source Code : LALR Bottom Up Parser

<parser.l>

```
%{
#include<stdio.h> #include "y.tab.h"
%}
%%
[0-9]+ {yylval.dval=atof(yytext); return DIGIT;
}
\n|. return yytext[0];
%%
```

<parser.y>

```
%{
/*This YACC specification file generates the LALR parser for the program considered in experiment
#include<stdio.h>
%}
%union
{
double dval;
}
%token <dval> DIGIT
%type <dval> expr
%type <dval> term
%type <dval> factor
%%
line: expr '\n' { printf("%g\n",$1);
}
;
expr: expr '+' term {$$=$1 + $3 ;}
| term
;
term: term '*' factor {$$=$1 * $3 ;}
| factor
;
factor: '(' expr ')' {$$=$2 ;}
| DIGIT
;
%%
int main()
{
yyparse();
}
yyerror(char *s)
{
printf("%s",s);
}
```

Output:

```
$lex parser.l
$yacc –d parser.y
$cc lex.yy.cy.tab.c –ll –lm
$./a.out 2+3
```

Viva Questions:

1. Define LALR parser
2. What is lookahead
3. What is augmented grammer
4. What are the most powerful LR parser
5. What is the difference between CLR and LALR

HOD **PRINCIPAL**