PRACTICAL-1

```
%{
#include<stdio.h>
#include<string.h>
%}

int 1-0;

/* Rules Section */

%%

([a-z])* (printf("%s is lowercase\n", yytext);}
([A-Z])* (printf("%s is capital letter\n",yytext);}
([0-9])* (printf("%s is number\n",yytext);}
([+1-x111@#$1^18]) (printf("%s is symbol/sign\n",yytext);}

%%

int yywrap(void){}

int main()
{
{printf("Enter your string: ");}

yylex();

return 0;
}
```

PRACTICAL-2

```
#include
<stdio.h>
char ch;
int fail
= 0;
void
next(char n) {
if
(ch
== n) {
ch =
getchar();
} else
{
printf("Error");
fail = 1;
}
}
void
next2() {
if
(ch == 'i') {
next('i');
next('*');
next2();} else if (ch == 't') {
next('t');
}
}
void start() {
next2();
if (ch == '+') {
next('+');
next('i');
}
else
{
fail
= 1;
printf("Error");
}
}
int
main() {
ch = getchar();
```

```
start();
if (!fail)
printf("Success")
; }
```

PRACTICAL-3

```
#include
<map>
#include
<set>
#include
<string>
#include
<vector>
using namespace std;
int
terminalCount;
int
nonTerminalCount;
int
productionCount;
map<string,
vector<string>> ps_map;
map<string,
set<string>>
FIRST;
map<string,
set<string>>
FOLLOW;
string
*ts;
string
*non_ts;
string
ss;
string
*ps;
template <typename T> set<T> getUnion(const set<T> &a, const set<T> &b)
{ set<T> result = a;
result.insert(b.begin(), b.end());
return result;
}
void c_p_c() {
#ifndef ONLINE_JUDGE
freopen("input.txt", "r", stdin);
freopen("o.txt", "w", stdout);
```

```
#endif
}
string getString(char x) {
string
s(1, x);
return
s;}
vector<string>
sp(string input, string delimiter)
{ size_t pos
=
0;
string token;
vector<string>
prods;
while ((pos
= input.find(delimiter)) != string::npos)
{ token =
input.substr(0, pos);
prods.push_back(token);
input.erase(0, pos + delimiter.length());
}
prods.push_back(input);
return prods;
}
bool inArray(string s, string *array, int size)
{ for (int i =
0; i < size; i++) {
if (array[i]
== s)
return true;
}
return false;
}
set<string> first(string s) {
using namespace std;
set<string> first_;
if (inArray(s, non_ts, nonTerminalCount))
{
vector<string> alternatives = ps_map[s];
for
(int i
= 0; i
< alternatives.size(); ++i)
{
string
```

```
temp =
alternatives[i];
set<string> first_2 = first(temp);
first_ = getUnion(first_, first_2);
}
}
else if (inArray(s, ts, terminalCount))
{ first_ =
{s};
}
else if (s
== "" ||
s == "@") {
first_ = {"@"};
}
else {
set<string> first_2
=
first(getString(s[0])); if
(first_2.find("@") != first_2.end()) {
int i = 1;while (first_2.find("@") != first_2.end())
{ set<string> ne = first_2;
ne.erase("@");
first_ = getUnion(first_, ne);
if (inArray(s.substr(i), ts, terminalCount))
{ set<string> t = {s.substr(i)};
first_ = getUnion(first_, t);
break;
} else if (s.substr(i) == "") {
set<string> t = {"@"};
first_ = getUnion(first_, t);
break;
}
ne = first(s.substr(i));
ne.erase("@");
first_
= getUnion(first_, ne);
i++;
}
} else {
first_ =
getUnion(first_, first_2);
}
}
return first_;
}
```

```
set<string> follow(string nT) {
using namespace std;
set<string> follow_;
if (nT == ss) {
set<string> dollar = {"$"};
follow_ =
getUnion(follow_, dollar);
}
map<string,
vector<string>>::iterator itr;
for
(itr =
ps_map.begin(); itr != ps_map.end(); ++itr)
{
string
nt = itr->first;
vector<string> rhs = itr->second;
for
(auto alt = rhs.begin(); alt != rhs.end(); ++alt)
{
for (int i = 0; i < (*alt).length(); i++) {
if (nT == getString((*alt)[i])) {string following_str
= (*alt).substr(i + 1);
if (following_str ==
"") {
if (nT == nt) {
continue;
} else {
follow_ =
getUnion(follow_, follow(nt));
}
} else {
set<string>
follow_2 =
first(following_str); if
(follow_2.find("@") != follow_2.end()) {
set<string> t = follow_2;
t.erase("@");
follow_
=
getUnion(follow_, t);
follow_
=
getUnion(follow_, follow(nt));
} else {
follow_
```

```
=
getUnion(follow_, follow_2);
}
}
}
}
}
}
return follow_;
}
int main() {
c_p_c();
cout << "Enter no. of ts: ";
cin >> terminalCount;
ts = new string[terminalCount];
cout << "Enter the ts :" << endl;
for (int
i = 0; i < terminalCount; i++) {
cin >>
ts[i];
}
// Non ts
cout << "Enter no. of non ts: ";
cin >>
nonTerminalCount;
non_ts
= new string[nonTerminalCount];cout << "Enter the non ts :" << endl;
for
(int i
= 0; i < nonTerminalCount; i++)
{
cin
>>
non_ts[i];
}
cout <<
"Enter the starting symbol: ";
cin >> ss;
cout << "Enter the number of ps: ";
cin >> productionCount;
ps = new string[productionCount];
cout << "Enter the ps: ";
for (int
i = 0; i < productionCount; i++) {
cin >>
ps[i];
```

```
vector<string>
temp = sp(ps[i], "->");
vector<string>
temp2 = sp(temp[1], "|");
ps_map.insert(pair<string, vector<string>>(temp[0],
temp2)); }
map<string, vector<string>>::iterator itr;
for
(itr
= ps_map.begin(); itr !=
ps_map.end(); ++itr)
{
cout
<< itr->first << " -> ";
vector<string>::iterator i;
for
(i =
itr->second.begin(); i
!= itr->second.end(); ++i)
{
cout
<< *i <<
" ";
}
cout << endl;
}
for (int i = 0; i <
nonTerminalCount; i++) {
FIRST[non_ts[i]] = {};
FOLLOW[non_ts[i]]
= {};
}
for (int i = 0; i <
nonTerminalCount; i++) {
FIRST[non_ts[i]]
=
getUnion(FIRST[non_ts[i]],
first(non_ts[i]));
}
set<string> dollar
=
{"$"};
FOLLOW[ss] = getUnion(FOLLOW[ss], dollar);for (int i = 0; i <
nonTerminalCount; i++) {
FOLLOW[non_ts[i]]
= getUnion(FOLLOW[non_ts[i]],
```

```
follow(non_ts[i]));
}
cout << "Non ts \t First \t\tFollow" << endl;
for (int i = 0; i <
nonTerminalCount; i++) {
cout << non_ts[i]
<< " \t\t ";
for (auto itr =
FIRST[non_ts[i]].begin(); itr != FIRST[non_ts[i]].end();
++itr) {
cout << *itr
<< " ";
}
cout << "\t\t";
for (auto itr = FOLLOW[non_ts[i]].begin(); itr !=
FOLLOW[non_ts[i]].end(); ++itr) {
cout << *itr << " ";
}
cout
<< endl;
}
return
0;
}
```

PRACTICAL-4

```
#include<stdio.h>
#include<string.h>
void main() {
char input[100],left[50],right[50],temp[10];
char productions[25][50];
int i=0,j=0,flag=0,consumed=0;
printf("#########################\t");
printf("Enter the production\t");
printf("#########################\n");
scanf("%1s->%s",left,right);
while(sscanf(right+consumed,"%[^|]s",temp) == 1 && consumed <= strlen(right))
{
if(temp[0] == left[0])
{
flag = 1;
sprintf(productions[i++],"%s'->%s%s'\0",left,temp+1,left);
}
else
{
sprintf(productions[i++],"%s->%s%s'\0",left,temp,left);
}
consumed += strlen(temp)+1;
}
if(flag == 1)
{
sprintf(productions[i++],"%s'->@\0",left);
printf("\n######################\t");
printf("The productions after removing left recursion are:\t");
printf("#########################\n");
for(j=0;j<i;j++)
{printf("%s\n",productions[j]);
}
}
else
printf("The given grammar has no left recursion");
}
```

PRACTICAL-5

calculator.l

```
%{
/* Definition Section*/
/*Lex Definition for Calculator*/
#include <stdio.h>
#include "y.tab.h"
extern int yylval;
%}
/*Rule Section*/
%%
\-?[0-9]+ {yylval = atoi(yytext);
return NUMBER;}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

Calculator.y

```
/*Parser Definition for Calculator*/
%{
/*Definition Section*/
#include <stdio.h>
int flag=0;
%}
/*Tokens and Operator Precedence*/
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
/*Rule Section*/
%%
/*Starting Symbol - Expression*/
Expression:E{
printf("\nResult = %d\n",$$);
return 0;
}
/*Context Free Grammar*/
```

```
E:E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|'('E')' {$$=$2;}
| NUMBER {$$=$1;}
;
%%
//Driver Code to Accept user input
void main()
{
while(1)
{
printf("\nEnter Expression\n");
yyparse();
if(!flag)
printf("\nExpression Valid\n");
}
}
void yyerror()
{
printf("\nExpression Invalid\n");flag = 1;
}
```

```
C:\Users\pc\Desktop\courses\bda>flex calculator.l

C:\Users\pc\Desktop\courses\bda>gcc lex.yy.c y.tab.c -w

C:\Users\pc\Desktop\courses\bda>a.exe

Enter Expression
5+6

Result = 11

Expression Valid

Enter Expression
1+8_

Result = 9

Expression Valid

Enter Expression

Expression Invalid

Enter Expression
1+(5*8)

Result = 41

Enter Expression
((6+9)

Expression Invalid

Enter Expression
```

PRACTICAL-6

```
#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.assignment\n2.arithmetic\n3.Exit\nEnter the choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the expression with assignment operator:");scanf("%s",exp);
l=strlen(exp);
exp2[0]='\0';
i=0;
while(exp[i]!='=')
{
i++;
}
strncat(exp2,exp,i);
strrev(exp);
exp1[0]='\0';
strncat(exp1,exp,l-(i+1));
strrev(exp1);
printf("Three address code:\ntemp=%s\n%s=temp\n",exp1,exp2);
break;
case 2:
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 3:
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]);
}
```

```
1.assignment
2.arithmetic
3.Exit
Enter the choice:1

Enter the expression with assignment operator:x=y
Three address code:
temp=y
x=temp

1.assignment
2.arithmetic
3.Exit
Enter the choice:1

Enter the expression with assignment operator:a*b=c
Three address code:
temp=c
a*b=temp

1.assignment
2.arithmetic
3.Exit
Enter the choice:2

Enter the expression with arithmetic operator:a+b-c
Three address code:
temp=a+b
temp1=temp-c

1.assignment
2.arithmetic
3.Exit
Enter the choice:2

Enter the expression with arithmetic operator:x/y-z
Three address code:
temp=x/y
temp1=temp-z

1.assignment
2.arithmetic
3.Exit
Enter the choice:3
```

PRACTICAL-7

Code:

control_stmt.y

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token ID NUM IF THEN LE GE EQ NE OR AND ELSE WHILE FOR DO
%right '='
%left AND OR
%left '<' '>' LE GE EQ NE
%left '+"-'
%left '*"/'
%right UMINUS
%left '!'
%%
S : ST {printf("Input accepted.\n");exit(0);};
ST : IF '(' E2 ')' THEN ST1';' ELSE ST1';'
| IF '(' E2 ')' THEN ST1';'
| WHILE '(' E2 ')' '{' ST1 ';' '}'
| FOR '(' E ';' E2 ';' E ')' '{' ST1 ';' '}'
| DO '{' ST1 ';' '}' WHILE '(' E2 ')'
;
ST1 : ST
| E
;
E : ID'='E
| E'+'E
| E'-'E
| E'*'E
| E'/'E
| E'<'E
| E'>'E
| E LE E
| E GE E
| E EQ E
| E NE E
| E OR E
| E AND E
| ID
| NUM;
E2 : E'<'E
| E'>'E
```

```
| E LE E
| E GE E
| E EQ E
| E NE E
| E OR E
| E AND E
| ID
| NUM
;
%%
main()
{
return(yyparse());
}
yyerror(s)
char *s;
{
fprintf(stderr, "%s\n",s);
}
yywrap()
{
return(1);
}
```

Control_stmt.l

```
%{
#include <stdio.h>
#include "y.tab.h"
%}
alpha [A-Za-z]
digit [0-9]
%%
[ \t\n]
if return IF;
then return THEN;
else return ELSE;
while return WHILE;
for return FOR;
do return DO;{digit}+ return NUM;
{alpha}({alpha}|{digit})* return ID;
"<=" return LE;
">=" return GE;
"==" return EQ;
"!=" return NE;
```

```
"||" return OR;
"&&" return AND;
. return yytext[0];
%%
```

PRACTICAL-8

Code:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int n,i,k,flag=0;
char vari[15],typ[15],b[15],c;
printf("Enter the number of variables:");
scanf(" %d",&n);
for(i=0;i<n;i++)
{
printf("Enter the variable[%d]:",i);
scanf(" %c",&vari[i]);
printf("Enter the variable-type[%d](float-f,int-i):",i);
scanf(" %c",&typ[i]);
if(typ[i]=='f')
flag=1;
}
printf("Enter the Expression(end with $):");
i=0;
getchar();
while((c=getchar())!='$')
{
b[i]=c;
i++; }
k=i;
for(i=0;i<k;i++)
{
if(b[i]=='/')
{
flag=1;
break; } }
for(i=0;i<n;i++)
{
if(b[0]==vari[i])
{
if(flag==1)
{
if(typ[i]=='f')
{ printf("\nthe datatype is correctly defined..!\n");
break; }
else{ printf("Identifier %c must be a float type..!\n",vari[i]);
```

```
break; } }
else
{ printf("\nthe datatype is correctly defined..!\n");
break; } }
}
return 0;
}
```

PRACTICAL-9

Code:

```
#include <stdio.h>
#include <stdio.h>
#include<conio.h>
#include <string.h>
void main() {
char icode[10][30], str[20], opr[10];
int i = 0;
printf("\n Enter the set of intermediate code (terminated by exit):\n");
do
{
scanf("%s", icode[i]);
} while (strcmp(icode[i++], "exit") != 0);
printf("\n target code generation");
printf("\n**********");
i = 0;
do {
strcpy(str, icode[i]);
switch (str[3]) {
case '+':strcpy(opr, "ADD ");
break;
case '-':
strcpy(opr, "SUB ");
break;
case '*':
strcpy(opr, "MUL ");
break;
case '/':
strcpy(opr, "DIV ");
break;
}
printf("\n\tMov %c,R%d", str[2], i);
printf("\n\t%s%c,R%d", opr, str[4], i);
printf("\n\tMov R%d,%c", i, str[0]);
} while (strcmp(icode[++i], "exit") != 0);
getch();
}
```

PRACTICAL-10

Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct op
{ char l;
char r[20];
}
op[10],pr[10];void main()
{ int a,i,k,j,n,z=0,m,q;
char *p,*l;
char temp,t;
char *tem;
printf("Enter the Number of Values:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("left: ");
op[i].l=getche();
printf("\tright: ");
scanf("%s",op[i].r);
}
printf("Intermediate Code\n") ;
for(i=0;i<n;i++)
{
printf("%c=",op[i].l);
printf("%s\n",op[i].r);
}
for(i=0;i<n-1;i++)
{ temp=op[i].l;
for(j=0;j<n;j++)
{
p=strchr(op[j].r,temp);
if(p)
{
pr[z].l=op[i].l;
strcpy(pr[z].r,op[i].r);
z++; }}}
pr[z].l=op[n-1].l;
strcpy(pr[z].r,op[n-1].r);
z++;
printf("After Dead Code Elimination\n");
```

```
for(k=0;k<z;k++) {
printf("%c\t=",pr[k].l);
printf("%s\n",pr[k].r);
}
for(m=0;m<z;m++) {
tem=pr[m].r;
for(j=m+1;j<z;j++) {
p=strstr(tem,pr[j].r);
if(p) {
t=pr[j].l;pr[j].l=pr[m].l;
for(i=0;i<z;i++) {
l=strchr(pr[i].r,t) ;
if(l) {
a=l-pr[i].r;
printf("pos: %d\n",a);
pr[i].r[a]=pr[m].l; }}}}}
printf("Eliminate Common Expression\n");
for(i=0;i<z;i++)
{
printf("%c\t=",pr[i].l);
printf("%s\n",pr[i].r);
}
for(i=0;i<z;i++)
{
for(j=i+1;j<z;j++)
{
q=strcmp(pr[i].r,pr[j].r);
if((pr[i].l==pr[j].l)&&!q)
{
pr[i].l='\0';
strcpy(pr[i].r,'\0');
}}}
printf("Optimized Code\n");
for(i=0;i<z;i++)
{ if(pr[i].l!='\0')
{
printf("%c=",pr[i].l);
printf("%s\n",pr[i].r);
}
}
getch();
}
```
