

LP ASSIGNMENT – 2

Rallapalle Kumar Chowdary

CSE-C

Roll no: 207259

1.

Implement CYK algorithm for deciding membership of a string in CFG in CNF. Assume that terminals and non-terminals are represented by a single alphabet and each grammar rule is given as a string where the first symbol is left side and the remaining portion is the corresponding right side of the production. Modify it to produce the number of non-identical derivation sequences for a given string in the grammar. Also generate the distinct derivation sequences, if possible.

Code:

```
#include<bits/stdc++.h>
using namespace std;
```

```

unordered_map < string, vector < string >> mp;
    //map of string, vector<string>
void
print ()
{
// printing lhs string
    for (auto it = mp.begin (); it != mp.end (); ++it)
    {
// cout<<it->first<<" -> ";
        for (int f = 0; f != it->second.size (); ++f)
        {
            cout << it->second[f];
        }
        cout << endl;
    }
}

```

```

int
main ()
{
//input productions
//no of productions is stored in n
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i)
    {

```

```

    string lhs, rhs;
    cout << "enter LHS ";
    cin >> lhs;
    cout << "enter RHS ";
    cin >> rhs;
    mp.insert (make_pair (rhs, vector < string >
    ()));
    mp[rhs].push_back (lhs);
}
print ();
string inp;
cout << "Enter string to verify grammer of :";
cin >> inp;
int len = inp.length ();
// cout<<inp<<endl;
//initialise cell len x len and fill diagonal
//making a 2D matrix of len X len
//filling the diagonals
// this is the format of our matrix
// a
// b
// b
// b
// a b b b
vector < string > dp[len][len];
for (int i = 0; i < len; ++i)

```

```

    {
        string s = "";
        s += inp[i];
        dp[i][i] = mp[s];
    }
//Fill other cells
// l=2 se start kiya hai to len tak pura jayega
for (int l = 2; l <= len; ++l)
{
    for (int i = 0; i < len - l + 1; ++i)
    {
        int j = i + l - 1;
        vector < string > res;
        for (int k = i; k < j; ++k)
        {
//dp (i,k)
            vector < string > v = dp[i][k];
//dp (k+1,j)
            vector < string > v1 = dp[k + 1][j];
//making the combos
            for (int p = 0; p < v.size (); ++p)
            {
                for (int m = 0; m < v1.size (); ++m)
                {
                    string combo = "";
                    combo += (v[p] + v1[m]);

```

```

//if present in productions
    if (mp.find (combo) != mp.end ())
    {
        for (int q = 0; q < mp[combo].size ();
++q)
            {
                res.push_back (mp[combo][q]);
            }
        }
    }
    dp[i][j] = res;
}
}
int flag = 0;
for (int i = 0; i < dp[0][len - 1].size (); ++i)
{
    if (dp[0][len - 1][i] == "S")
    {
        cout << "IN rules" << endl;
        flag = 1;
        break;
    }
}
if (flag == 0)

```

```
    cout << "out of rules" << endl;
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            vector < string > v = dp[i][j];
            for (auto it:v)
            {
                cout << it;
            }
            cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:

```

rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2$ g++ -o cyk cyk.cpp
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2$ ./cyk
8
enter LHS S
enter RHS AB
enter LHS S
enter RHS BC
enter LHS A
enter RHS BA
enter LHS A
enter RHS a
enter LHS B
enter RHS CC
enter LHS B
enter RHS b
enter LHS C
enter RHS AB
enter LHS C
enter RHS a
B
AC
A
S
B
SC
enter string to verify grammer of :baaaba
out of rules
B AS SASC SSSCSC BB
AC B SCAS SCSSC BBB
AC B B SCAAS
AC SC B
B AS
AC
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2$

```

2.

Write flex specifications to convert an infix expression to postfix notation by constructing the corresponding expression tree. Use the binary operators +, -, *, /, @(exponentiation) and um (Unary Minus). Assume usual precedence and associativity rules for

the operators. Note that some of the operator symbols are meta-characters in flex.

//unary minus is used to show a negative number.

Code:

```
%{
    #include<stdio.h>
    #include<stdlib.h>
    typedef struct node{
        char *data;
        struct node *left;
        struct node *right;
    }*tptr;
    char *charstack[100];
    tptr nodestack[100];
    int topnode=-1,topchar=-1;
    int precedence(char);
}%
%%
"um"|\@|[\+\-\*\^\(\)\%] {
                                if(topchar== -1)
                                {
                                    charstack[+
+topchar]=(char *)malloc(strlen(yytext)+1);

                                strcat(charstack[topchar],yytext);
                                }
                                else
                                {
                                    if(yytext[0]=='(')
                                    {
```



```

charstack[+
+topchar]=(char *)malloc(strlen(yytext)+1);

    strcat(charstack[topchar],yytext);
    }
    else if(yytext[0]==')')
    {

        while(charstack[topchar][0]!='(')
            {
                tptr
temp=(tptr)malloc(sizeof(tptr));
                temp-
>data=(char *)malloc(strlen(charstack[topchar])+1);
                strcat(temp-
>data,charstack[topchar]);

                if(topnode<1)
                {

                    printf("Wrong Input\n");

                    return -1;
                }
                temp-
>right=nodestack[topnode];

                topnode--;
                if(yytext[0]=='u')
                    temp-
>left=NULL;

                else
                {
                    temp-
>left=nodestack[topnode];

                    topnode--;
                }

```

```

+topnode]=temp;

printf("Wrong Input\n");

nodestack[+
topchar--;
if(topchar==-1)
{
return -1;
}
}
topchar--;
}
else
{
if(yytext[0]=='@'||
yytext[0]=='u')
{
while(topchar>=0&&precedence(yytext[0])<precedence(charstack
[topchar][0]))
{
tptr
temp=(tptr)malloc(sizeof(tptr));
temp-
>data=(char *)malloc(strlen(charstack[topchar])+1);
if(topnode<1)
{
printf("Wrong Input\n");
return -
1;
}

```

```

>data,charstack[topchar]);
>right=nodestack[topnode];

    if(yytext[0]=='u')
>left=NULL;

    >left=nodestack[topnode];

    topnode--;

    +topnode]=temp;

while(topchar>=0&&precedence(yytext[0])<=precedence(charstack[topchar][0]))

temp=(tptr)malloc(sizeof(tptr));
>data=(char *)malloc(strlen(charstack[topchar])+1);

    if(topnode<1)

```

```

strcat(temp-
temp-
topnode--;

temp-
else
{
temp-
}
nodestack[+
topchar--;
}
else
{
tptr
temp-
{

```

```

    printf("Wrong Input\n");

    return -
1;
    }
    strcat(temp-
>data,charstack[topchar]);
    temp-
>right=nodestack[topnode];
    topnode--;

    if(yytext[0]=='u')
        temp-
>left=NULL;
    else
    {
        temp-
>left=nodestack[topnode];

        topnode--;

    }
    nodestack[+
+topnode]=temp;
    topchar--;
}
}
charstack[+
+topchar]=(char *)malloc(strlen(yytext)+1);

    strcat(charstack[topchar],yytext);
    }
}
}
[a-z]+|[a-z]*[0-9]+ {
    tptr temp=(tptr)malloc(sizeof(tptr));

```

```

temp->data=(char
*)malloc(strlen(yytext)+1);
strcat(temp->data,yytext);
temp->left=NULL;
temp->right=NULL;
nodestack[++topnode]=temp;
    }

[' \t\n] {;}
%%
int precedence(char c)
{
    if(c=='+'||c=='-')
        return 1;
    else if(c=='*'||c=='/')
        return 2;
    else if(c=='u')
        return 3;
    else if(c=='@')
        return 4;
    else if(c=='(')
        return 0;
}
void prefix(tptr T)
{
    if(T==NULL) return;
    printf("%s",T->data);
    prefix(T->left);
    prefix(T->right);
}
void infix(tptr T)
{
    if(T==NULL) return;
    infix(T->left);
    printf("%s",T->data);
    infix(T->right);
}

```

```

}
void postfix(tptr T)
{
    if(T==NULL) return;
    postfix(T->left);
    postfix(T->right);
    printf("%s",T->data);
}
int main()
{
    printf("Enter the input expression : ");
    yylex();
    while(topchar>=0)
    {
        tptr temp=(tptr)malloc(sizeof(tptr));
        temp->data=(char *)malloc(strlen(charstack[topchar]
+1));
        if(topnode<1)
        {
            printf("Wrong Input\n");
            return -1;
        }
        strcat(temp->data,charstack[topchar]);
        temp->right=nodestack[topnode];
        topnode--;
        if(yytext[0]=='u')
            temp->left=NULL;
        else
        {
            temp->left=nodestack[topnode];
            topnode--;
        }
        nodestack[++topnode]=temp;
        topchar--;
    }
}

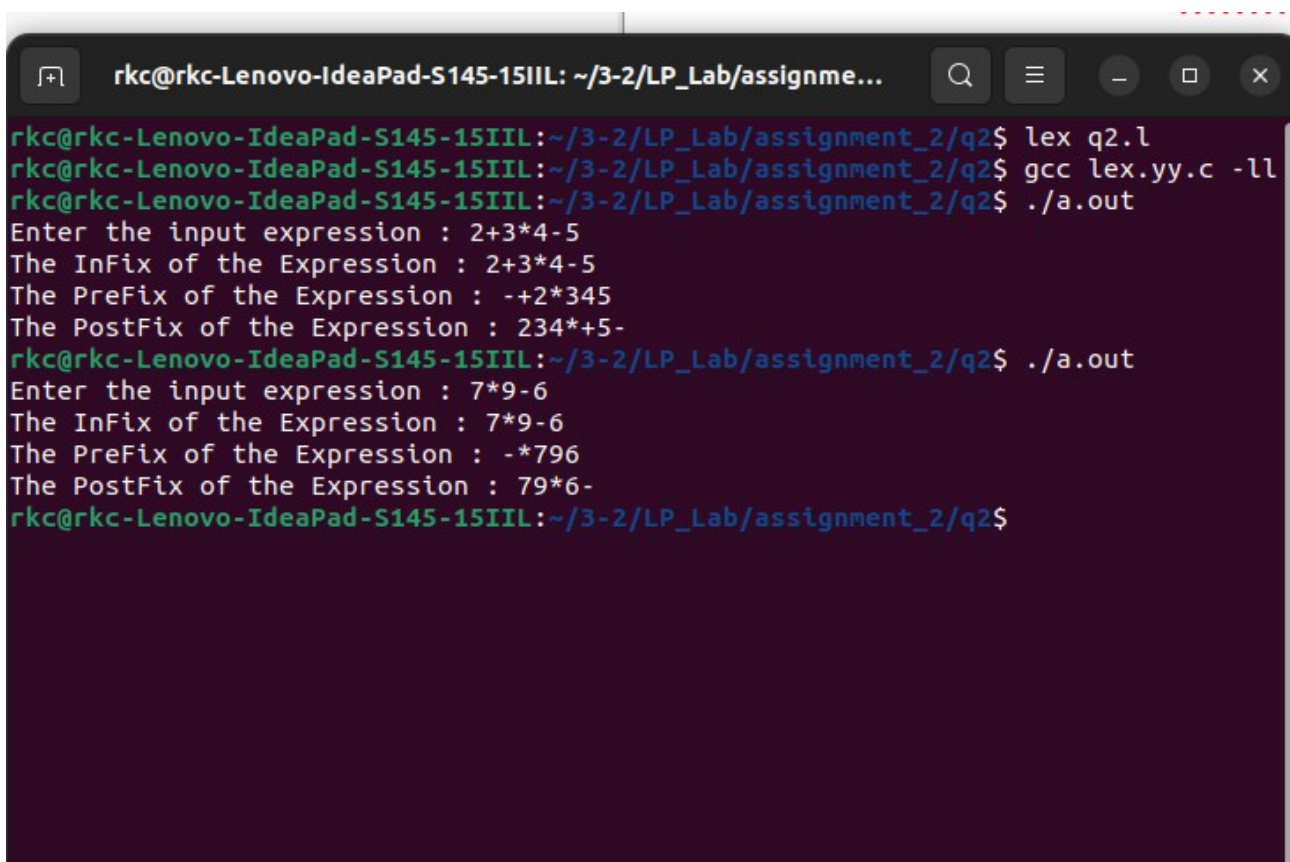
```

```

    tptr expressiontree=nodestack[topnode];
    printf("The InFix of the Expression : ");
    infix(expressiontree);
    printf("\n");
    printf("The PreFix of the Expression : ");
    prefix(expressiontree);
    printf("\n");
    printf("The PostFix of the Expression : ");
    postfix(expressiontree);
    printf("\n");
    return 0;
}

```

Output:



```

rkc@rkc-Lenovo-IdeaPad-S145-15IIL: ~/3-2/LP_Lab/assignme...
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q2$ lex q2.l
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q2$ gcc lex.yy.c -ll
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q2$ ./a.out
Enter the input expression : 2+3*4-5
The InFix of the Expression : 2+3*4-5
The PreFix of the Expression : -+2*345
The PostFix of the Expression : 234*+5-
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q2$ ./a.out
Enter the input expression : 7*9-6
The InFix of the Expression : 7*9-6
The PreFix of the Expression : -*796
The PostFix of the Expression : 79*6-
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q2$

```

Q3

Consider a CFG with each terminal and non-terminal represented as single alphabet and each production is represented as a string with left most symbol as left side variable of the production. Write a program to eliminate useless symbols, ϵ - productions and unit productions.

From the resultant grammar remove immediate left recursion. Display the input and output grammars in a readable notation.

Code:

```
#include<bits/stdc++.h>
using namespace std;
set<string> epsilonproductions(map<string,vector<string>>
G)
{
    set<string> oldvariables,newvariables;
    do
    {
        oldvariables=newvariables;
        for(auto i : G)
```



```

{
    for(auto j : i.second)
    {
        if(j=="@")
        {
            newvariables.insert(i.first);
            break;
        }
        else
        {
            for(int k=0;k<j.length();k++)
            {
                string temp;
                temp.push_back(j[k]);

if(oldvariables.find(temp)==oldvariables.end())
                    break;
                    if(k==j.length()-1)
                        newvariables.insert(i.first);
            }
        }
    }
}while(oldvariables!=newvariables);
return newvariables;
}
map<string,vector<string>>
removeepsilonproductions(map<string,vector<string>>
&G,set<string> epsilonvariables)
{
    map<string,vector<string>> G1;

```

```

do
{
    G1=G;
    for(auto i : G)
    {
        for(auto j : i.second)
        {
            for(int k=0;k<j.length();k++)
            {
                string temp;
                temp.push_back(j[k]);
                if(epsilonvariables.find(temp)!
=epsilonvariables.end())
                {
                    string s1=j.substr(0,k)
+j.substr(k+1,j.length()-k-1);
                    for(int p=0;p<G[i.first].size();p++)
                    {
                        if(G[i.first][p]==s1)
                            break;
                        if(p==G[i.first].size()-1&& s1.size()>0)
                            G[i.first].push_back(s1);
                    }
                }
            }
        }
    }
    }while(G1!=G);
    G1.clear();
    for(auto i : G)
    {

```

```

        for(auto j : i.second)
        {
            if(j!="@")
                G1[i.first].push_back(j);
        }
    }
    return G1;
}

void removingunitproductions(map<string,vector<string>>
&G)
{
    set<string> S;
    for(auto i : G)
    {
        for(auto j : i.second)
        {
            if(j.length()==1&&j[0]>='A'&&j[0]<='Z')
            {
                S.insert(i.first);
            }
        }
    }
    map<string,vector<string>> G1;
    do
    {
        G1=G;
        for(auto i : S)
        {
            for(int j=0;j<G[i].size();j++)
            {

```

```

        if(G[i][j].length()==1&&G[i][j][0]>='A'&&G[i]
[j][0]<='Z')
        {
            string c=G[i][j];
            G[i].erase(G[i].begin()+j);
            for(auto p : G[c])
            {
                bool present=true;
                for(auto q : G[i])
                {
                    if(p==q||(p.length()==1&&p==i))
                    {
                        present=false;
                        break;
                    }
                }
                if(present) G[i].push_back(p);
            }
        }
    }
}while(G!=G1);
}
map<string,vector<string>>
removalofuselessproductions(map<string,vector<string>>
&G,string start)
{
    set<string> OV,NV;
    do
    {
        OV=NV;

```

```

for(auto i : G)
{
    for(auto j : i.second)
    {
        bool inserted=false;
        for(int p=0;p<j.length();p++)
        {
            string temp;
            temp.push_back(j[p]);

if(j[p]>='A'&& j[p]<='Z'&& NV.find(temp)==NV.end())
                break;
            if(p==j.length()-1)
            {
                NV.insert(i.first);
                inserted=true;
            }
        }
        if(inserted) break;
    }
}while(OV!=NV);
map<string,vector<string>> G1;
vector<string> traverse;
traverse.push_back(start);
set<string> pushed;
pushed.insert(start);
for(int i=0;i<traverse.size();i++)
{
    for(auto j : G[traverse[i]])
    {

```

```

        for(int p=0;p<j.length();p++)
        {
            string temp;
            temp.push_back(j[p]);

            if(j[p]>='A'&&j[p]<='Z'&&NV.find(temp)==NV.end())
                break;

            if(j[p]>='A'&&j[p]<='Z'&&pushed.find(temp)==pushed.end())
            {
                pushed.insert(temp);
                traverse.push_back(temp);
            }
            if(p==j.length()-1)
                G1[traverse[i]].push_back(j);
        }
    }
}
return G1;
}
map<string,vector<string>>
removingimmediateleftrecursion(map<string,vector<string>>> G)
{
    map<string,vector<string>> G1;
    set<string> S;
    for(auto i : G)
    {
        for(auto j : i.second)
        {

```

```

        if(j[0]==i.first[0])
        {
            string temp;
            temp.push_back(i.first[0]);
            S.insert(temp);
        }
    }
}
for(auto i : G)
{
    if(S.find(i.first)==S.end())
    {
        for(auto j : i.second)
            G1[i.first].push_back(j);
    }
    else
    {
        vector<string> alpha,beta;
        for(auto j : i.second)
        {
            if(i.first[0]==j[0])
                alpha.push_back(j.substr(1,j.length()-1));
            else
                beta.push_back(j);
        }
        for(auto j : beta)
            G1[i.first].push_back(j);
        for(auto j : beta)
            G1[i.first].push_back(j+i.first[0]+"");
        for(auto j : alpha)
            G1[i.first+""].push_back(j);
    }
}

```

```

        for(auto j : alpha)
            G1[i.first+"\"].push_back(j+i.first+"\"");
    }
}
return G1;
}
int main()
{
    map<string,vector<string>> G;
    cout<<"Enter the CFG (use @ for epsilon) (to terminate
enter #)\n";
    string s;
    getline(cin,s);
    string start;
    while(s!="#")
    {
        string left;
        string right;
        bool first=false;
        for(int i=0;i<s.length();i++)
        {
            if(s[i]!=' ' && !first && start.length()==0)
                start.push_back(s[i]);
            if(s[i]!=' ' && !first)
                left.push_back(s[i]),first=true;
            else if(s[i]!=' ')
                right.push_back(s[i]);
        }
        right=right.substr(2,right.length()-2);
        G[left].push_back(right);
        getline(cin,s);
    }
}

```



```

}
set<string> eplisonvariables=epsilonproductions(G);
if(eplisonvariables.size()>=1)
cout<<"Epsilon Productions : ";
for(auto i : eplisonvariables)
    cout<<i<<" ";
cout<<endl;
G=removeepsilonproductions(G,eplisonvariables);
cout<<"Grammer after removing epsilon productions\n";
for(auto i : G)
    for(auto j : i.second)
        cout<<i.first<<" -> "<<j<<endl;
cout<<"Grammer After removing unit productions\n";
removingunitproductions(G);
for(auto i : G)
    for(auto j : i.second)
        cout<<i.first<<" -> "<<j<<endl;
cout<<"Grammer After removing useless productions\n";
G=removalofuselessproductions(G,start);
for(auto i : G)
    for(auto j : i.second)
        cout<<i.first<<" -> "<<j<<endl;
cout<<"Grammer After removing immediate left
recursions\n";
G=removingimmediateleftrecursion(G);
for(auto i : G)
    for(auto j : i.second)
        cout<<i.first<<" -> "<<j<<endl;
return 0;
}

```

```
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2$ g++ grammar.cpp
```

```
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2$ ./a.out
```

```
Enter the CFG (use @ for epsilon) (to terminate enter #)
```

```
E->E+T
```

```
E->T
```

```
T->T*F
```

```
T->F
```

```
F->(E)
```

```
F->id
```

```
#
```

```
Grammar after removing epsilon productions
```

```
E -> E+T
```

```
E -> T
```

```
F -> (E)
```

```
F -> id
```

```
T -> T*F
```

```
T -> F
```

```
Grammar After removing unit productions
```

```
E -> E+T
```

```
E -> T*F
```

```
E -> (E)
```

```
E -> id
```

```
F -> (E)
```

```
F -> id
```

```
T -> T*F
```

```
T -> (E)
```

```
T -> id
```

```
Grammar After removing useless productions
```

```
E -> E+T
```

```
E -> T*F
```

```
E -> (E)
```

```
E -> id
```

```
F -> (E)
```

```
F -> id
```

```
T -> T*F
```

```
T -> (E)
```

```
T -> id
```

```
Grammar After removing immediate left recursions
```

```
E -> T*F
```

```
E -> (E)
```

```
E -> id
```

```
E -> T*FE'
```

```
E -> (E)E'
```

```
E -> idE'
```

```
E' -> +T
```

```
E' -> +TE'
```

```
F -> (E)
```

```
F -> id
```

```
T -> (E)
```

```
T -> id
```

```
T -> (E)T'
```

```
T -> idT'
```

```
T' -> *F
```

```
T' -> *FT'
```

```
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2$
```

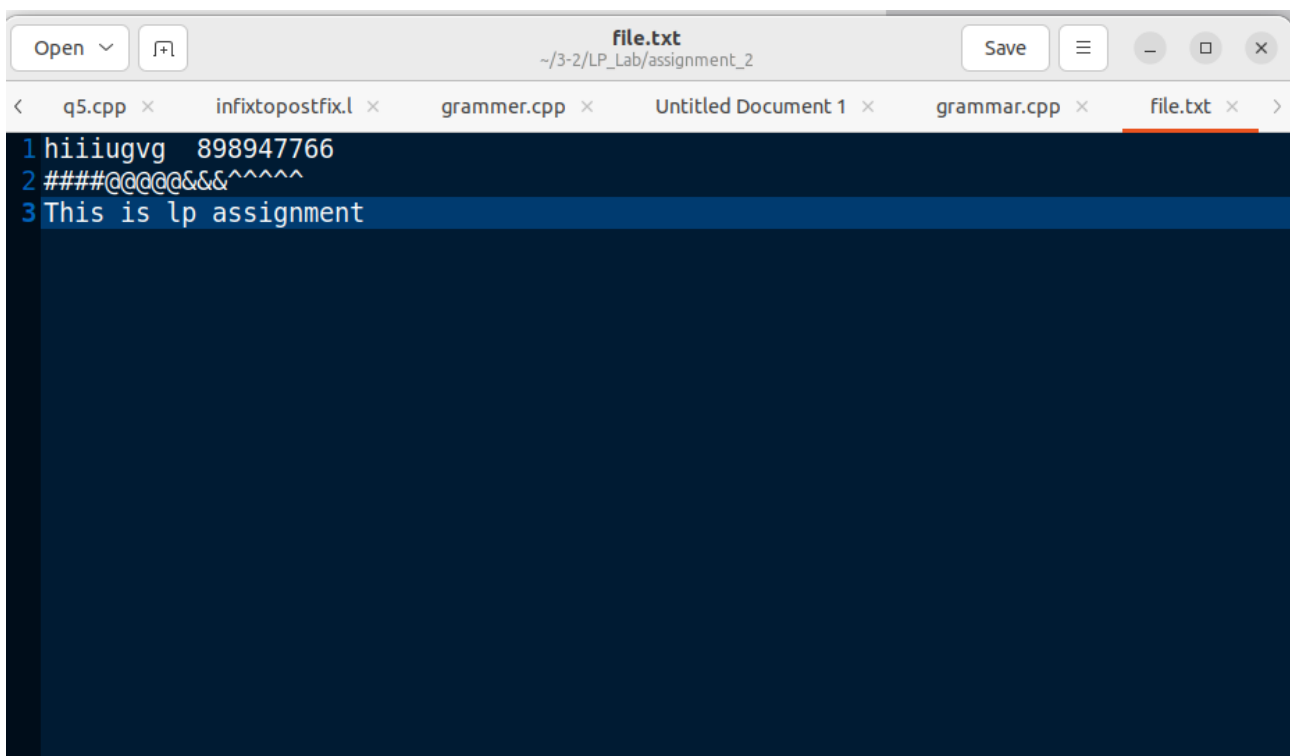
4.

Write a program to encrypt the text using lex. Copy a text file with some predefined words as special and encrypt the text as follows:

- Special words have to be rotated right by two positions (cyclic) and written right to left.
- Other English words should be encrypted by replacing each character by ASCII + k-cyclic and words need to be written in reverse.
- Each number should be replaced by a new number which is generated by swapping left half and right half of the number. If the

number has odd number of digits, the middle digit should be at the same place

- All other characters have to be retained as it is except the sequence of whitespaces which need to be replaced by single space.



The screenshot shows a code editor window with a tab titled 'file.txt' at the path '~/3-2/LP_Lab/assignment_2'. The editor contains the following text:

```
1 hiiugvg 898947766
2 ####@@@@&&^^^^^
3 This is lp assignment
```

```

1 %{
2 #include <stdio.h>
3 #include <string.h>
4 #include <ctype.h>
5 char* rotate_special(char* word);
6 char* encrypt_word(char* word);
7 char* swap_number(char* number);
8 char* reverse(char* str);
9
10 %}
11
12 %option noyywrap
13
14 %%
15 [ \t\r\n]+          { printf(" "); }
16 [0-9]+              { printf("%s", swap_number(yytext)); }
17 [a-zA-Z]+          { printf("%s", encrypt_word(yytext)); }
18 .                  { printf("%s", rotate_special(yytext)); }
19
20
21 %%
22
23
24 char* reverse(char *str) {
25     int i, j;
26     char tmp;
27     for (i = 0, j = strlen(str) - 1; i < j; i++, j--) {
28         tmp = str[i];
29         str[i] = str[j];
30         str[j] = tmp;
31     }
32     return str;
33 }
34
35 char* rotate_special(char* word) {
36     int len = strlen(word);
37     char* rotated = (char*)malloc(len + 1);
38     int i;
39
40     for (i = 0; i < len - 2; i++) {
41         rotated[i] = word[i+2];
42     }
43
44     rotated[len-2] = word[0];
45     rotated[len-1] = word[1];
46     rotated[len] = '\0';
47
48     return reverse(rotated);
49 }
50

```

```
50
51 char* encrypt_word(char* word) {
52     int len = strlen(word);
53     char* encrypted = (char*)malloc(len + 1);
54     int i;
55
56     for (i = 0; i < len; i++) {
57         encrypted[i] = word[len-i-1] + 3;
58     }
59
60     encrypted[len] = '\0';
61
62     return encrypted;
63 }
64
65 char* swap_number(char* number) {
66     int len = strlen(number);
67     char* swapped = (char*)malloc(len + 1);
68     int i, mid;
69
70     if (len % 2 == 0) {
71         mid = len / 2;
72     } else {
73         mid = len / 2 + 1;
74     }
75
76     for (i = 0; i < len; i++) {
77         if (i < mid) {
78             swapped[i] = number[i+mid];
79         } else {
80             swapped[i] = number[i-mid];
81         }
82     }
83
84     swapped[len] = '\0';
85
86     return swapped;
87 }
88
```

```

88
89 int main(int argc, char** argv) {
90     FILE* file;
91
92     if (argc != 2) {
93         printf("Usage: %s <filename>\n", argv[0]);
94         return 1;
95     }
96
97     file = fopen(argv[1], "r");
98
99     if (!file) {
100         printf("Error opening file.\n");
101         return 1;
102     }
103
104     yyin = file;
105     yylex();
106
107     fclose(file);
108
109     return 0;
110 }

```

Output: after encryption

```

rkc@rkc-Lenovo-IdeaPad-S145-15IIL: ~/3-2/LP_Lab/assignment_2/q4
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q4$ lex q4.l
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q4$ gcc lex.yy.c -ll
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q4$ ./a.out file.txt
rrrrrr if4y84f76 9u74gy8gug ugf33tf

```

5.

Implement a desk calculator with the operators defined in assignment 2 along with named identifiers, assignment statements, if statement, logical expressions (<, >, ==, != for less than, greater than, equal to and not equal to as used in C language). Write a function "let" for initializing a variable and "display" to display the value of a variable or expression.

Lex part:

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include "parser.h"  
%}  
%%  
"+" { return ADD; }  
"-" { return SUB; }  
"*" { return MUL; }  
"/" { return DIV; }  
%" { return MOD; }  
"<" { return LT; }  
">" { return GT; }
```



```

"<=" { return LE; }
">=" { return GE; }
"==" { return EQ; }
"!=" { return NE; }
"=" { return ASSIGN; }
"if" { return IF; }
"else" { return ELSE; }
"let" { return LET; }
"display" { return DISPLAY; }
[0-9]+ { yylval.number = atof(yytext); return NUMBER; }
[a-zA-Z][a-zA-Z0-9]* { yylval.name = yytext; return
NAME; }
[ \t\n] { /* ignore whitespace */ }
. { printf("Unknown character: %s\n", yytext); exit(1); }
%%
int yywrap(void) {
return 1;
}

```

Yacc Part:

```

%{
#include <stdio.h>
#include <stdlib.h>
#include "parser.h"
#include "lexer.h"
#include "symbol_table.h"
%}
%token ADD SUB MUL DIV MOD LT GT LE GE EQ NE
ASSIGN IF ELSE LET

```

DISPLAY

%token <number> NUMBER

%token <name> NAME

%left ADD SUB

%left MUL DIV MOD

%right ASSIGN

%nonassoc LT GT LE GE EQ NE

%%

program:

/* empty */

| program statement

;

statement:

expression ';' { display(\$1); }

| LET NAME '=' expression ';' { let(\$2, \$4); }

| IF '(' expression ')' statement ELSE statement

{ if_statement(\$3, \$5, \$7); }

;

expression:

NUMBER { \$\$ = \$1; }

| NAME { \$\$ = get_value(\$1); }

| expression ADD expression { \$\$ = \$1 + \$3; }

| expression SUB expression { \$\$ = \$1 - \$3; }

| expression MUL expression { \$\$ = \$1 * \$3; }

| expression DIV expression { \$\$ = \$1 / \$3; }

| expression MOD expression { \$\$ = (int)\$1 % (int)\$3; }

| expression LT expression { \$\$ = \$1 < \$3; }

| expression GT expression { \$\$ = \$1 > \$3; }

| expression LE expression { \$\$ = \$1 <= \$3; }

| expression GE expression { \$\$ = \$1 >= \$3; }

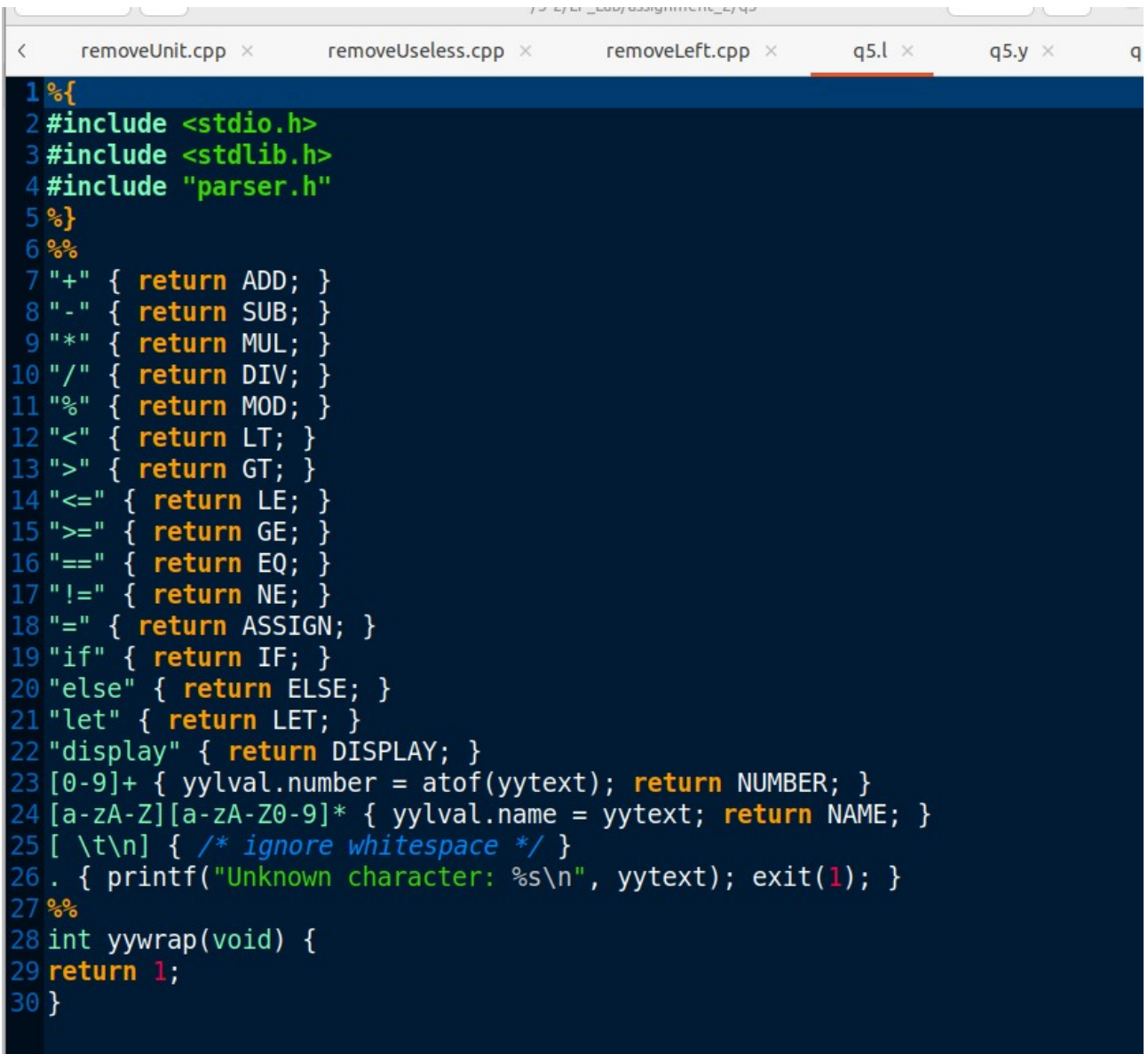
| expression EQ expression { \$\$ = \$1 == \$3; }

```

| expression NE expression { $$ = $1 != $3; }
;
%%
int main() {
  yyparse();
  return 0;
}
void yyerror(const char* s) {
  fprintf(stderr, "%s\n", s);
}
void let(char* name, double value) {
  set_value(name, value);
}
double get_value(char* name) {
  double value = lookup_value(name);
  if (value == SYMBOL_NOT_FOUND) {
    fprintf(stderr, "Error: Undefined variable '%s'\n", name);
    exit(EXIT_FAILURE);
  }
  return value;
}
void if_statement(double condition, Node* if_statement,
Node* else_statement){
  if (condition){
    execute(if_statement);
  }
  else {
    execute(else_statement);
  }
}
void display(double value) {

```

```
printf("%.2f\n", value);  
}
```



```
1 %{  
2 #include <stdio.h>  
3 #include <stdlib.h>  
4 #include "parser.h"  
5 %}  
6 %%  
7 "+" { return ADD; }  
8 "-" { return SUB; }  
9 "*" { return MUL; }  
10 "/" { return DIV; }  
11 "%" { return MOD; }  
12 "<" { return LT; }  
13 ">" { return GT; }  
14 "<=" { return LE; }  
15 ">=" { return GE; }  
16 "==" { return EQ; }  
17 "!=" { return NE; }  
18 "=" { return ASSIGN; }  
19 "if" { return IF; }  
20 "else" { return ELSE; }  
21 "let" { return LET; }  
22 "display" { return DISPLAY; }  
23 [0-9]+ { yylval.number = atof(yytext); return NUMBER; }  
24 [a-zA-Z][a-zA-Z0-9]* { yylval.name = yytext; return NAME; }  
25 [ \t\n] { /* ignore whitespace */ }  
26 . { printf("Unknown character: %s\n", yytext); exit(1); }  
27 %%  
28 int yywrap(void) {  
29     return 1;  
30 }
```

Open ▾



q5.y

~/3-2/LP_Lab/assignment_2/

cyk.cpp ×

q2.l ×

q2.y ×

removeNull.cpp ×

removeUnit.cpp ×

rem

```
1 %{
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "parser.h"
5 #include "lexer.h"
6 #include "symbol_table.h"
7 %}
8 %token ADD SUB MUL DIV MOD LT GT LE GE EQ NE ASSIGN IF ELSE LET
9 DISPLAY
10 %token <number> NUMBER
11 %token <name> NAME
12 %left ADD SUB
13 %left MUL DIV MOD
14 %right ASSIGN
15 %nonassoc LT GT LE GE EQ NE
16 %%
17 program:
18 /* empty */
19 | program statement
20 ;
21 statement:
22 expression ';' { display($1); }
23 | LET NAME '=' expression ';' { let($2, $4); }
24 | IF '(' expression ')' statement ELSE statement { if_statement($3, $5, $7); }
25 ;
26 expression:
27 NUMBER { $$ = $1; }
28 | NAME { $$ = get_value($1); }
29 | expression ADD expression { $$ = $1 + $3; }
30 | expression SUB expression { $$ = $1 - $3; }
31 | expression MUL expression { $$ = $1 * $3; }
32 | expression DIV expression { $$ = $1 / $3; }
33 | expression MOD expression { $$ = (int)$1 % (int)$3; }
34 | expression LT expression { $$ = $1 < $3; }
35 | expression GT expression { $$ = $1 > $3; }
36 | expression LE expression { $$ = $1 <= $3; }
37 | expression GE expression { $$ = $1 >= $3; }
38 | expression EQ expression { $$ = $1 == $3; }
39 | expression NE expression { $$ = $1 != $3; }
40 ;
41 %%
```

```

42 int main() {
43 yyparse();
44 return 0;
45 }
46 void yyerror(const char* s) {
47 fprintf(stderr, "%s\n", s);
48 }
49 void let(char* name, double value) {
50 set_value(name, value);
51 }
52 double get_value(char* name) {
53 double value = lookup_value(name);
54 if (value == SYMBOL_NOT_FOUND) {
55 fprintf(stderr, "Error: Undefined variable '%s'\n", name);
56 exit(EXIT_FAILURE);
57 }
58 return value;
59 }
60 void if_statement(double condition, Node* if_statement, Node* else_statement){
61 if (condition){
62 execute(if_statement);
63 }
64 else {
65 execute(else_statement);
66 }
67 }
68 void display(double value) {
69 printf("%.2f\n", value);
70 }

```

Cpp Code:

```

#include <iostream>
#include <map>
#include <string>
#include <sstream>
// Store variables and their values
std::map < std::string, int >
    variables;
// Function to assign a value to a variable
void
let (std::string variable, int value)
{
    variables[variable] = value;
}

```

```

// Function to evaluate an expression and return the result
int
eval (std::string expression)
{
    std::istringstream ss (expression);
    int
        lhs,
        rhs;
    std::string op;
    // If the expression is a variable name, return its value
    if (variables.count (expression))
    {
        return variables[expression];
    }
    // If the expression is a binary expression (e.g. 1 + 2)
    ss >> lhs >> op >> rhs;
    if (op == "+")
    {
        return lhs + rhs;
    }
    else if (op == "-")
    {
        return lhs - rhs;
    }
    else if (op == "*")
    {
        return lhs * rhs;
    }
    else if (op == "/")
    {
        return lhs / rhs;
    }
}

```

```

    }
    // If the expression is not a variable or binary expression,
    return 0
    return 0;
}

// Function to display the value of a variable or expression
void
display (std::string expression)
{
    std::cout << eval (expression) << std::endl;
}

int
main ()
{
    let ("a", 10);
    display ("a");
    let ("b", 20);
    display ("a + b");
    return 0;
}

```

Output:



rkc@rkc-Lenovo-Ide

```
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2$ cd q5
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q5$ g++ q5.cpp
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q5$ ./a.out
10
0
rkc@rkc-Lenovo-IdeaPad-S145-15IIL:~/3-2/LP_Lab/assignment_2/q5$
```