Question 1: Expression Validity Checker

C++ Source Code

```
// Q1: Expression Validity Checker
Q1.l :
%option noyywrap
%{
#include <stdio.h>
#include "Q1.tab.h"
%}
%%
[ \t\r\n]+                ;
[a-zA-Z][a-zA-Z0-9]*      { return
VARIABLE; }
[0-9]+                    { return NUMBER;
}
"="                       { return '='; }
"("                       { return '('; }
")"                       { return ')'; }
"+"                       { return '+'; }
"-"                       { return '-'; }
"*"                       { return '*'; }
"/"                       { return '/'; }
"%"                       { return '%'; }
.                      { return yytext[0]; }
%%
```

```
Q1.y :
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(const char *s);
int yylex(void);
%}
%token NUMBER VARIABLE
%left '+' '-'
%left '*' '/' '%'
%%
input:
      stmt
    ;
stmt:
      VARIABLE '=' expr   {
printf("\naccepted\n\n"); exit(0); }
    | expr                {
printf("\naccepted (expression)\n\n");
exit(0); } ;
expr:
      expr '+' expr
    | expr '-' expr
    | expr '*' expr
    | expr '/' expr
    | expr '%' expr
    | '(' expr ')'
    | NUMBER
    | VARIABLE;
%%
int main(void){
    printf("\nEnter Any Arithmetic
Expression:\n");
    yyparse();
    return 0;
}
void yyerror(const char *s){
    printf("\nrejected\n\n");
    exit(0);
}
```

Sample Input / Output

```
Input: id + id * ( id - id )
Output: accepted
```

Question 2: Expression Evaluator

C++ Source Code

```
// Q2: Expression Evaluator
Q2.l :
%option noyywrap
%{
#include <stdio.h>
#include "Q2.tab.h"
extern long long yylval;
%}
%%
[ \t\r\n]+                ;
-?[0-9]+                  {
yylval = atoll(yytext);
return NUMBER;}
"("                       { return '('; }
")"                       { return ')'; }
"+"                       { return '+'; }
"-"                       { return '-'; }
"*"                       { return '*'; }
"/"                       { return '/'; }
.                         {return
yytext[0]; } %%
```

```
Q2.y :
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(const char *s);
int yylex(void);
extern long long yylval;
%}
%union {
    long long ival;
}
%token <ival> NUMBER
%type <ival> expr
%left '+' '-'
%left '*' '/'
%right UMINUS
%%
input:
      expr    { printf("\nResult =
%lld\n", $1); exit(0); };
expr:
      expr '+' expr   { $$ = $1 + $3; }
    | expr '-' expr   { $$ = $1 - $3; }
    | expr '*' expr   { $$ = $1 * $3; }
    | expr '/' expr   { if ($3 == 0)
yyerror("division by zero");
       $$ = $1 / $3; }
    | '-' expr %prec UMINUS  { $$ = -$2; }
    | '(' expr ')'     { $$ = $2; }
    | NUMBER           { $$ = $1; } ;
%%
int main(void) {
    printf("Enter arithmetic
expression:\n");
    fflush(stdout);
    yyparse();
    return 0;
}
void yyerror(const char *s) {
    printf("\nError: %s\n", s);
    exit(0);
}
```

Sample Input / Output

```
Input: ( 2 + 3 ) * 4 - 6 / 2
Output: 18
```

Question 3: Remove Left Recursion

C++ Source Code

```
// Q3: Remove Immediate Left Recursion
(C++)

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

int main(){
    cout<<"Enter production (example: A ->
A a | b):\n";
    string line; getline(cin,line);

    auto trim=[&](string s){
        while(!s.empty() &&
isspace(s.front())) s.erase(s.begin());
        while(!s.empty() &&
isspace(s.back())) s.pop_back();
        return s;
    };

    auto pos=line.find("->");
    string A=trim(line.substr(0,pos));
    string rhs=trim(line.substr(pos+2));

    vector<string> prods;
    string temp="";
    for(char c:rhs){
        if(c=='|'){
prods.push_back(trim(temp)); temp=""; }
        else temp+=c;
    }
    if(temp.size())
prods.push_back(trim(temp));

    vector<string> alpha, beta;

    for(auto &p:prods){
        stringstream ss(p); string first;
ss>>first;
        if(first==A){
            string rest; getline(ss,rest);
            if(rest.size() && rest[0]=='
') rest=rest.substr(1);
            alpha.push_back(rest);
        }else beta.push_back(p);
    }

    if(alpha.empty()){
```

```
        cout<<A<<" -> ";
        for(int i=0;i<prods.size();i++){
            if(i) cout<<" | ";
            cout<<prods[i];
        }
    }else{
        string Aprime=A+"'";
        cout<<A<<" -> ";
        for(int i=0;i<beta.size();i++){
            if(i) cout<<" | ";
            cout<<beta[i]<<" "<<Aprime;
        }
        cout<<"\n"<<Aprime<<" -> ";
        for(int i=0;i<alpha.size();i++){
            if(i) cout<<" | ";
            cout<<alpha[i]<<" "<<Aprime;
        }
        cout<<" | epsilon";
    }
}
```

Sample Input / Output

```
Input:
A -> A a | b

Output:
A -> b A'
A' -> a A' | epsilon
```

Question 4: Left Factoring

C++ Source Code

```
// Q4: Left Factoring (C++)

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

int main(){
    cout<<"Enter production (A -> a b | a
c | d): ";
    string line; getline(cin,line);

    auto trim=[&](string s){
        while(!s.empty() &&
isspace(s.front())) s.erase(s.begin());
        while(!s.empty() &&
isspace(s.back())) s.pop_back();
        return s;
    };

    auto pos=line.find("->");
    string A=trim(line.substr(0,pos));
    string rhs=trim(line.substr(pos+2));

    vector<string> prods;
    string temp="";
    for(char c:rhs){
        if(c=='|'){
prods.push_back(trim(temp)); temp=""; }
        else temp+=c;
    }
    if(temp.size())
prods.push_back(trim(temp));

    // detect common prefix
    vector<vector<string>> tok_prods;
    for(auto &p:prods){
        stringstream ss(p); string x;
vector<string> v;
        while(ss>>x) v.push_back(x);
        tok_prods.push_back(v);
    }

    string prefix = tok_prods[0][0];
    bool common=true;
    for(auto &p:tok_prods){
        if(p[0]!=prefix) common=false;
    }

    if(!common){
        cout<<line;
        return 0;
    }

    string Aprime=A+"_a";
    cout<<A<<" -> "<<prefix<<"
"<<Aprime<<" | d
";
    cout<<Aprime<<" -> b | c";
}
```

Sample Input / Output

```
Input:
A -> a b | a c | d

Output:
A -> a A_a | d
```

```
A_a -> b | c
```

Question 5: FIRST & FOLLOW

C++ Source Code

```
// Q5: FIRST & FOLLOW (C++)

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

int main(){
    cout<<"Enter grammar (line by line,
blank to stop):\n";
    map<string, vector<vector<string>>> G;
    string line;

    auto trim=[&](string s){

while(!s.empty()&&isspace(s.front()))
s.erase(s.begin());

while(!s.empty()&&isspace(s.back()))
s.pop_back();
        return s;
    };

    while(true){
        getline(cin,line);
        if(line=="") break;
        auto pos=line.find("->");
        string L=trim(line.substr(0,pos));
        string
rhs=trim(line.substr(pos+2));

        string temp="";
        vector<string> prods;
        for(char c:rhs){
            if(c=='|'){
prods.push_back(trim(temp)); temp=""; }
            else temp+=c;
        }
        if(temp!="")
prods.push_back(trim(temp));

        for(auto &p:prods){
            vector<string> tokens;
stringstream ss(p); string t;
            while(ss>>t)
tokens.push_back(t);
            G[L].push_back(tokens);
        }
    }

    map<string,set<string>> FIRST, FOLLOW;

    function<void(string)> calcFIRST =
[&](string X){
        for(auto &prod:G[X]){
            if(prod[0]=="epsilon"){
FIRST[X].insert("epsilon"); continue; }
            for(auto &sym:prod){
                if(!G.count(sym)){
FIRST[X].insert(sym); break; }
                calcFIRST(sym);
                for(auto &t:FIRST[sym])
if(t!="epsilon") FIRST[X].insert(t);

if(!FIRST[sym].count("epsilon")) break;
            }
        }
    };

    for(auto &p:G) calcFIRST(p.first);

    string start = G.begin()->first;
    FOLLOW[start].insert("$");

    bool changed=true;
    while(changed){
        changed=false;
        for(auto &A:G){
            for(auto &prod:A.second){
                for(int
i=0;i<prod.size();i++){
                    string B=prod[i];
                    if(!G.count(B))
continue;
                    set<string> trailer;
                    if(i+1<prod.size()){
                        for(int
j=i+1;j<prod.size();j++){
                            string
sym=prod[j];

if(!G.count(sym)){ trailer.insert(sym);
break;}
                            for(auto
&t:FIRST[sym]) if(t!="epsilon")
trailer.insert(t);
```

```
if(!FIRST[sym].count("epsilon")) break;

if(j==prod.size()-1)
                                for(auto
&t:FOLLOW[A.first]) trailer.insert(t);
                        }
                    }else{
                        for(auto
&t:FOLLOW[A.first]) trailer.insert(t);
                    }
                    int
prev=FOLLOW[B].size();

FOLLOW[B].insert(trailer.begin(),
trailer.end());

if(FOLLOW[B].size()!=prev) changed=true;
                }
            }
        }
    }

    cout<<"
FIRST:
";
    for(auto &p:FIRST){
        cout<<p.first<<" : ";
        for(auto &t:p.second) cout<<t<<"
";
        cout<<"
";
    }

    cout<<"
FOLLOW:
";
    for(auto &p:FOLLOW){
        cout<<p.first<<" : ";
        for(auto &t:p.second) cout<<t<<"
";
        cout<<"
";
    }
}
```

Sample Input / Output

```
Input:
E -> T E'
E' -> + T E' | epsilon
T -> F T'
T' -> * F T' | epsilon
F -> ( E ) | id

FIRST:
E : ( id
E' : + epsilon
T : ( id
T' : * epsilon
F : ( id

FOLLOW:
E : ) $
E' : ) $
T : + ) $
T' : + ) $
F : * + ) $
```

Question 6: Recursive Descent Parser

C++ Source Code

```
// Q6: Recursive Descent Parser (accepts
ep)

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

string pk(const vector<string>& t,int p){
return p<t.size()?t[p]:"$"; }

bool mt(vector<string>& t, int &p, string
s){
    if(pk(t,p)==s){ p++; return true; }
    return false;
}

// Grammar:
// E -> T E'
// E'-> + T E' | ep
// T -> F T'
// T'-> * F T' | ep
// F -> ( E ) | id

bool E(const vector<string>&, int&);
bool E_(const vector<string>&, int&);
bool T(const vector<string>&, int&);
bool T_(const vector<string>&, int&);
bool F(const vector<string>&, int&);
```

```
bool F(const vector<string>& tokens, int
&p){
    if(pk(tokens,p)=="("){
        p++;
        if(!E(tokens,p)) return false;
        if(pk(tokens,p)!=")") return
false;
        p++;
        return true;
    }
    if(pk(tokens,p)=="id"){
        p++;
        return true;
    }
    return false;
}

bool T_(const vector<string>& tokens, int
&p){
    if(pk(tokens,p)=="ep"){ p++; return
true; }
    if(pk(tokens,p)=="*"){ p++; return
F(tokens,p)&&T_(tokens,p); }
    return true;
}

bool T(const vector<string>& tokens, int
&p){
    return F(tokens,p)&&T_(tokens,p);
}

bool E_(const vector<string>& tokens, int
&p){
    if(pk(tokens,p)=="ep"){ p++; return
true; }
    if(pk(tokens,p)=="+"){ p++; return
T(tokens,p)&&E_(tokens,p); }
    return true;
}

bool E(const vector<string>& tokens, int
&p){
    return T(tokens,p)&&E_(tokens,p);
}

int main(){
    cout<<"Enter tokens: ";
    string line; getline(cin,line);
    stringstream ss(line); vector<string>
tokens; string t;
    while(ss>>t) tokens.push_back(t);
    int p=0;
    bool ok = E(tokens,p) &&
p==tokens.size();
    cout<<(ok?"accepted":"rejected");
}
```

Sample Input / Output

```
Input:
( id + id ) * id
Output:
accepted
```

Question 7: LL(1) Parser Simulation

C++ Source Code

```
// Q7: LL(1) Parser Simulation (formatted)

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

void printStack(const vector<string>& st){
    for(auto &s:st) cout<<s<<" ";
}

void printInput(const vector<string>&
in,int ip){
    for(int i=ip;i<in.size();i++)
        cout<<in[i]<<" ";
}

int main(){
    cout<<"Enter tokens: ";
    string line; getline(cin,line);
    stringstream ss(line); vector<string>
input; string tok;
    while(ss>>tok) input.push_back(tok);
    input.push_back("$");

    vector<string> stack={"$","E"};
    int ip=0;

    cout<<"Stack\t\tInput\t\tAction\n";

    while(!stack.empty()){
        string top=stack.back();
        string cur=input[ip];
```

```
        printStack(stack); cout<<"\t\t";
        printInput(input,ip);
cout<<"\t\t";

        if(top==cur && top=="$"){
            cout<<"Accept"; break;
        }

        if(top==cur){
            stack.pop_back(); ip++;
            cout<<"Match "<<cur<<"\n";
            continue;
        }

        if(top=="E"){
            if(cur=="id"||cur=="("){
                stack.pop_back();
                stack.push_back("E'");
                stack.push_back("T");
                cout<<"E -> T E'\n";
                continue;
            }
        }

        if(top=="E'"){
            if(cur=="+"){
                stack.pop_back();
                stack.push_back("E'");
                stack.push_back("T");
                stack.push_back("+");
                cout<<"E' -> + T E'\n";
                continue;
            }
            stack.pop_back(); cout<<"E' ->
epsilon\n"; continue;
        }

        if(top=="T"){
            if(cur=="id"||cur=="("){
                stack.pop_back();
                stack.push_back("T'");
                stack.push_back("F");
                cout<<"T -> F T'\n";
                continue;
            }
        }

        if(top=="T'"){
            if(cur=="*"){
                stack.pop_back();
                stack.push_back("T'");
                stack.push_back("F");
                stack.push_back("*");
                cout<<"T' -> * F T'\n";
                continue;
            }
            stack.pop_back(); cout<<"T' ->
epsilon\n"; continue;
        }

        if(top=="F"){
            if(cur=="id"){
                stack.pop_back();
                stack.push_back("id");
                cout<<"F -> id\n";
                continue;
            }
            if(cur=="("){
                stack.pop_back();
                stack.push_back(")");
                stack.push_back("E");
                stack.push_back("(");
                cout<<"F -> ( E )\n";
                continue;
            }
        }

        cout<<"Error";
        break;
    }
}
```

Sample Input / Output

Input:

```
id + id * id
```

Output:

```
$ E          id + id * id $       E -> T E
```