University *of* Alexandria

Faculty of Engineering

Computer and Systems Engineering Department

Compilers

Phase 1

Lexical Analyzer

Names:

Ahmed Ibrahim Hafez (04)

Ahmed El-Sayed Mahmoud (05)

Amr Gamal Mohamed (45)

Mohamed El-Sayed Mohamed (54)

- ***Data Structures***

Graph

The graph consists of states and edges

```
#ifndef GRAPH_H_INCLUDED
#define GRAPH_H_INCLUDED
#include <stack>

#include "State.h"

using namespace std;

class Graph
{
public:
    Graph();

    void set_start_state(State* start);
    State* get_start_state();

    void set_end_state(State* endd);
    State* get_end_state();

    void set_graph_size(int s);
    int get_graph_size();
    void clear_visited();

    virtual ~Graph();

private:
    State * start;
    State * endd;
    int graph_size =0 ;
};

#endif // GRAPH_H_INCLUDED
```

```cpp
#include "Edge.h"

using namespace std;

class State
{
public:
    State();

    void set_state_num(int num);
    int get_state_num();

    void set_accepting(bool acceptance, string accepted_token);
    bool is_accepting_state();
    string get_accepted_token();

    void add_child_state(State* to, string weight);

    bool get_visit();
    void set_visit(bool v);

    vector<Edge> * get_children();

    virtual ~State();

private:
    bool visited = false;
    bool is_accepting = false;
    string accepted_token = "";
    int state_num = -1;
    vector<Edge> children;
};
```

```cpp
#ifndef EDGE_H_INCLUDED
#define EDGE_H_INCLUDED

class State;

using namespace std;

class Edge
{
public:
    Edge(State* from, State* to, string weight);

    pair<State*, State*> get_start_end_states();
    State* getTo();
    string get_weight();

    virtual ~Edge();

private:
    string weight;
    State *from;
    State *to;
};


#endif // EDGE_H_INCLUDED
```

Grammar parser & NFA construction

```
#include "Graph.h"

class GrammerParser
{
public:
    GrammerParser(string grammerFile);
    Graph* getFullNFA();
    Graph* clone(Graph* g);
    vector<string> get_weights();
    vector<string> get_expressions();

    virtual ~GrammerParser();
private:
    string grammerFile;
    vector<string> grammer;
    int statenum = 0;
    unordered_map<string,Graph*> definitions;
    unordered_map<string,Graph*> expressions;
    vector<string> expNames;
    vector<string> weights;

    void getNFA(string line);
    void readGrammerFile(string path);
    void addOperation(char c, stack<string>* in, stack<string>* post);
    bool isOperation(char c);
    bool isOperation(string c);
    string removeSpaces(string str);
    vector<string> getTokens(string str2);
    stack<string> getPostfix(vector<string> tokens);
    bool isBinaryOp(string c);
    Graph* draw(stack<string> postfix, string acceptance);
    Graph* drawOR(Graph* g1, Graph* g2, string acceptance);
    Graph* drawCO(Graph* g1, Graph* g2, string acceptance);
    Graph* drawKclosure(Graph* g1, string acceptance);
    Graph* drawPclosure(Graph* g1, string acceptance);
    void printGraph(Graph* g);
    void setWeights(Graph* g);
    bool check_for_weight(string weight);
};

#endif // GRAMMERPARSER_H
```

NFA to DFA

We construct the DFA represented as table of DFA_State

```
#ifndef DFA_STATE_H_INCLUDED
#define DFA_STATE_H_INCLUDED

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

class DFA_State
{
public:

    DFA_State(int dfa_state_size );

    void add_nfa_state (int nfa_state_num, bool is_acceptance);
    void set_accepting(bool acceptance, string accepted_token);
    void set_state_num(int num );
    bool is_accepting_state ();
    int get_state_num();
    vector <bool> get_nfa_states_accept();
    vector <int> get_nfa_states_nums();
    string get_accepted_token();
    bool is_empty_state ();
    virtual ~DFA_State();
private:
    bool is_acceptance, is_empty_s;
    vector <bool> nfa_states;
    string accepted_token;
    int dfa_state_num;
    vector <int> nfa_states_nums;
};

#endif // DFA_STATE_H_INCLUDED
```

```
#include "Edge.h"
#include "DFA_State.h"
#include <bits/stdc++.h>

using namespace std;

class NFA_To_DFA
{
public:
//Map to hold all inputs and its mapping to column number in transition table
    NFA_To_DFA();
    NFA_To_DFA(Graph * NFA, vector<string> input, vector<string> tokens);
    vector < vector < DFA_State > > get_DFA_table ();
    vector < DFA_State> get_DFA_states ();
    vector<string> get_accepted_tokens();

    virtual ~NFA_To_DFA();

private:
    void DFA_BFS(State *s, unordered_map<string,int> input_map);
    void do_subset_construction (int numberOfInputs);
    int get_dfa_state_index (DFA_State state );
    void print_DFA_trans_table();
    vector < vector < int > > nfa_trans_table;
    vector < vector <int > > ep_clos_table;
    vector<bool> accepted_states;
    vector<string> accepted_tokens;
    vector < vector < DFA_State > > DFA_table ;
    vector <DFA_State> DFA_states ;
    vector <string> new_accepted_tokens ;
    unordered_map<string, int> acc_tokens_map;
};
```

Minimization

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

class Minimization
{

public:

    Minimization(unordered_map<string, int> tokensPriorities);
    vector < vector <DFA_State> > minimize_DFA (vector < vector <DFA_State> > DFA,vector <DFA_State> states );
    vector < DFA_State > get_minimum_states();
    virtual ~Minimization();

private:

    vector < vector <DFA_State> > min_DFA;
    vector < DFA_State > min_states;
    unordered_map<string, int> tokensPriorities;
    template<typename T> void printTable (T t, int width);
    void printTransitionTable (vector< vector <DFA_State> > states);

    ofstream tables;

};
```

Tokens

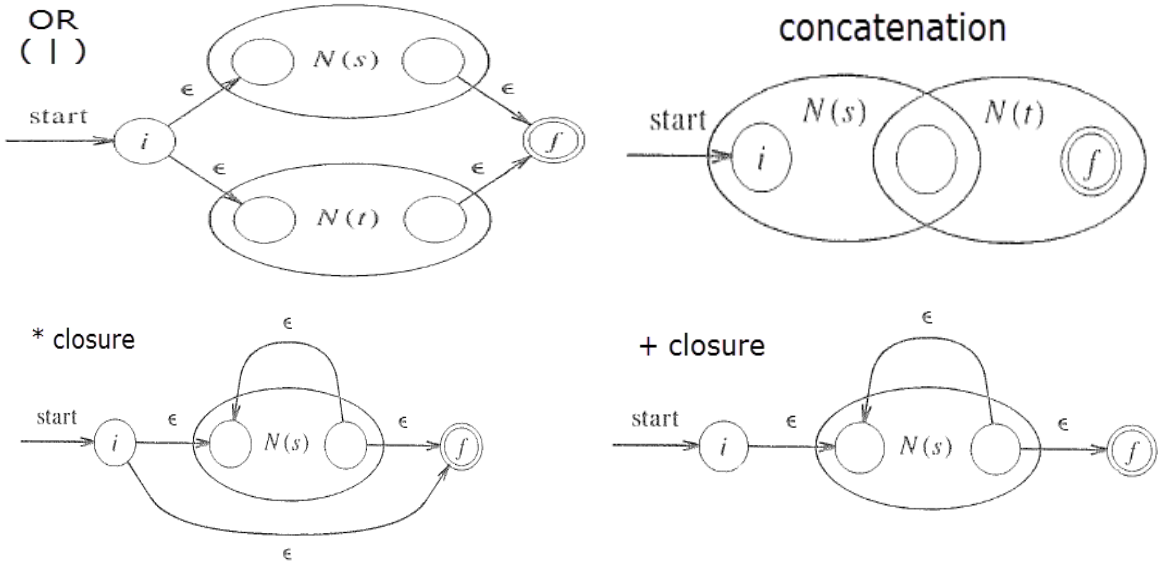
```
#include <string.h>
#include <stack>
#include <fstream>
#include "DFA_State.h"

using namespace std;

class Tokens
{
public:

    Tokens(vector <vector <DFA_State> > table, map <char, int> alphabet, vector <DFA_State> states,string infile);
    Tokens() {};

    string getToken();

    virtual ~Tokens();

private:

    char check_char();
    int get_alpha_idx(char test);
    ifstream inputFile;
    ofstream outputFile;
    ofstream errorFile;
    ofstream symbolFile;
    vector <vector <DFA_State> > transitionTable;
    map <char, int> alphabet;
    vector <DFA_State> states;
    vector <pair<DFA_State, char> > input;
    stack <pair<DFA_State, char> > readHistory;
    int currentStateNum;

};
```

- ***Algorithms & techniques***

Grammar parser & NFA construction

First the grammar file is read line by line and NFA of each line which will be a definition or key word or punctuation is drawn. The expression is drawn throw cloning some of the definitions that it needs. We used the same technique in making the | , + , * and concatenation operations as illustrated in the lecture.

Then all the drawn NFAs is combined together by a start node that is connected to them with an epsilon transition.



NFA to DFA

In this part we convert NFA to DFA by removing all epsilon edges using BFS. First we begin from the start node and save all nodes reached from epsilon edges in some data structure and make all this nodes in one new state (DFA_State) and then start from every node contained in this state and repeat the first step until all nodes in the NFA graph covered. The idea of this algorithm is "every group of nodes in NFA graph represent a single node in the DFA graph".

Minimization

First the code separates acceptance states -based on accepted string- from non-acceptance states then assign numbers to each set created.

The main loop iterates over the current set size then for each set member it's compared with all other elements if it matches transitions with other state it's removed from the set and added to a new set.
After creating the final minimal sets it takes the first element of each set as a representative for that set then it assigns new numbering for the minimized states.
Than iterate over the original DFA transition table replacing each state with its representative ignoring the states which are handled by their representatives then it returns the constructed minimized DFA table and minimized states.

Tokens

First it reads an input file character by character matching the read character with a state from transition table then proceed with the selected state.
It applies Maximal-Munch algorithm to handle any conflicts and applies the longest accepted token and print it to output file and the symbols to a symbol file, if any errors found due to non-accepting states or unknown characters it prints that symbol to error file.
It returns one token per every call to get token function.

- ***Minimum Transition Table***

The Minimum Transition Table is written in a file called table this file contains the transition table resulted from the DFA then the sets of states that are the same then the Minimum Transition Table.
Below is an rotated image of the Minimum Transition Table and the original image is provided in the zip file.

Minimized DFA table :

****

0	1	2	3	4	5	6	7	8	9	10	11	12	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	14	15	15	16	16	17	17	17	17	17	17	17	17	17	17	14	18	-1
1	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	19	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
2	13	13	20	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	21	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
3	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	22	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
4	-1	-1	-1	23	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	24	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
6	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	25	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
13	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
14	-1	-1	-1	23	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	17	17	17	17	17	17	17	17	17	17	-1	-1	26
18	-1	-1	-1	23	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
19	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	27	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
20	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
21	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	28	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
22	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	29	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
23	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
24	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	30	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
25	13	31	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
26	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	32	32	32	32	32	32	32	32	32	32	-1	-1	-1
27	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	33	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
28	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
29	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	34	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
30	13	13	13	-1	35	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
31	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	36	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
32	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	37	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	32	32	32	32	32	32	32	32	32	32	-1	-1	-1
33	13	13	13	-1	38	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	-1	-1	-1
34	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	39	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	-1	-1
35	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	-1	-1
36	13	13	13	-1	40	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	-1	-1
37	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	41	41	41	41	41	41	41	41	41	41	41	-1	-1
38	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	42	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	-1	-1
39	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	-1	-1
40	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	-1	-1
41	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	41	41	41	41	41	41	41	41	41	41	41	-1	-1
42	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	43	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	-1	-1
43	13	13	13	-1	13	13	-1	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	-1	-1	-1	-1	-1	13	13	13	13	13	13	13	13	13	13	13	-1	-1

- ***Tokens of test program***

Test Program

```
int sum , count , pass ,
mnt; while (pass != 10)
{
        pass = pass + 1 ;
}
```

Output tokens:

int
id
,
id
,
id
,
id
;
while
(
id
relop
num
)
{
id
assign
id
addop
num
;
}

- ***Assumptions***

Accepted tokens priority will be taken with the reversed order of Regular expressions file.