UCS802- COMPILER CONSTRUCTION

Lab Assignment – 1

Q1. Design a Minimized DFA for the Regular Expression (a/b)*abb i.e. All strings ending with abb.

```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <stack>
#include <map>
#include <cstring>
using namespace std;
#define COL 5
#define _S 30

// Setting the Global Variables
int init[20], fin[20], a = 0, b = 0;
string init_dfa[_S], fin_dfa[_S];
int _a = 0, _b = 0;

void initialise(int table_NFA[][COL]) {
    for(int i = 0; i < 1000; i++)
        for(int j = 0; j < COL; j++)
            table_NFA[i][j] = -1;
}

void print_initial_final() {
    cout << "Initial state/s is/are :- ";
    for(int i = 0; i < a; i++)
        cout << init[i] << " ";
    cout << endl;
    cout << "Final state/s is/are :- ";
    for(int i = 0; i < b; i++)
        cout << fin[i] << " ";
    cout << endl;
}

void print_initial_final_dfa() {
    cout << "Initial state/s is/are :- ";
    for(int i = 0; i < _a; i++)
        cout << init_dfa[i] << " ";
    cout << endl;
    cout << "Final state/s is/are :- ";
    for(int i = 0; i < _b; i++)
        cout << fin_dfa[i] << " ";
    cout << endl << endl;
    for (int i = 0; i < 60; i++)
        cout << "-";
    cout << endl << endl;
}

void reduce_fin(int x) {
    for(int i = x; i < b - 1; i++)
        fin[i] = fin[i + 1];
    b -= 1;
}

// Preprocessor Function
string preprocessor(string s) {
    char x[5000];
    auto l = s.length();
    int j = 0;
```

```
    x[j] = '(';
    j += 1;
    for(int i = 0; i < l; i++) {
        x[j] = s[i];
        j += 1;
        if(s[i] >= 97 && s[i] <= 122 && s[i + 1] >= 97 && s[i + 1] <= 122) {
            x[j] = '.';
            j += 1;
        }
        else if(s[i] == ')' && s[i + 1] == '(') {
            x[j] = '.';
            j += 1;
        }
        else if(s[i] >= 97 && s[i] <= 122 && s[i + 1] == '(') {
            x[j] = '.';
            j += 1;
        }
        else if(s[i] == ')' && s[i + 1] >= 97 && s[i + 1] <= 122) {
            x[j] = '.';
            j += 1;
        }
        else if(s[i] == '*' && (s[i + 1] == '(' || (s[i + 1] >= 97 && s[i + 1] < 122))) {
            x[j] = '.';
            j += 1;
        }
    }
    x[j] = ')';
    j += 1;
```

```
    string p;
    for(int i = 0; i < j; i++)
        p += x[i];
    return p;
}

// Postfix Function
string postfix(string s) {
    auto l = s.length();
    char a[5000];
    stack<char> ch;
    int j = 0;
    for(int i = 0; i < l; i++) {
        char x = s[i];
        switch(x) {
            case 'a':
                a[j] = 'a';
                j += 1;
                break;

            case 'b':
                a[j] = 'b';
                j += 1;
                break;

            case '(':
                ch.push('(');
                break;
```

```
115             case ')':
116                 while(!ch.empty()) {
117                     if(ch.top() == '(') {
118                         ch.pop();
119                         break;
120                     }
121                     else {
122                         a[j] = ch.top();
123                         ch.pop();
124                         j += 1;
125                     }
126                 }
127                 break;
128 
129             case '.':
130                 if(ch.empty())
131                     ch.push('.');
132                 else {
133                     char temp = ch.top();
134                     if(temp == '(')
135                         ch.push('.');
136                     else if(temp == '*') {
137                         a[j] = ch.top();
138                         ch.pop();
139                         j += 1;
140                         if(ch.top() == '.') {
141                             a[j] = '.';
142                             j += 1;
```

```
143                         }
144                         else
145                         ch.push('.');
146                     }
147                     else if(temp == '.') {
148                         a[j] = ch.top();
149                         ch.pop();
150                         j += 1;
151                         ch.push('.');
152                     }
153                     else if(temp == '|')
154                         ch.push('.');
155                 }
156                 break;
157 
158             case '|':
159                 if(ch.empty())
160                     ch.push('|');
161                 else {
162                     char temp = ch.top();
163                     if(temp == '(')
164                         ch.push('|');
165                     else if(temp == '*') {
166                     a[j] = ch.top();
167                     ch.pop();
168                     j += 1;
169                     ch.push('|');
170                     }
```

```
                else if(temp == '.') {
                    a[j] = ch.top();
                    j += 1;
                    ch.pop();
                    ch.push('|');
                }
            }
            break;

        case '*':
            if(ch.empty())
                ch.push('*');
            else {
                char temp = ch.top();
                if(temp == '(' || temp == '.' || temp == '|')
                    ch.push('*');
                else {
                    a[j] = ch.top();
                    ch.pop();
                    j += 1;
                    ch.push('*');
                }
            }
            break;
    }
}
string p;
for(int i = 0; i < j; i++)
    p += a[i];
return p;
}

// Regular Expression to NFA
int re_to_nfa(string s, int table_NFA[][COL]) {
    auto l = s.length();
    int states = 1;
    int m, n, j, count;
    for(int i = 0; i < l; i++) {
        char x = s[i];
        switch(x) {
            case 'a':
                table_NFA[states][0] = states;
                init[a] = states;
                a += 1;
                states += 1;
                table_NFA[states - 1][1] = states;
                fin[b] = states;
                b += 1;
                table_NFA[states][0] = states;
                states += 1;
                break;

            case 'b':
                table_NFA[states][0] = states;
                init[a] = states;
                a += 1;
```

```
            states += 1;
            table_NFA[states - 1][2] = states;
            fin[b] = states;
            b += 1;
            table_NFA[states][0] = states;
            states += 1;
            break;

        case '.':
            m = fin[b - 2];
            n = init[a - 1];
            table_NFA[m][3] = n;
            reduce_fin(b - 2);
            a -= 1;
            break;

        case '|':
            for(j = a - 1, count = 0; count < 2; count++) {
                m = init[j - count];
                table_NFA[states][3 + count] = m;
            }
            a = a - 2;
            init[a] = states;
            a += 1;
            table_NFA[states][0] = states;
            states += 1;
            for(j = b - 1, count = 0; count < 2; count++) {
                m = fin[j - count];
                table_NFA[m][3] = states;
            }
            b = b - 2;
            fin[b] = states;
            b += 1;
            table_NFA[states][0] = states;
            states += 1;
            break;

        case '*':
            m = init[a - 1];
            table_NFA[states][3] = m;
            table_NFA[states][0] = states;
            init[a - 1] = states;
            states += 1;
            n = fin[b - 1];
            table_NFA[n][3] = m;
            table_NFA[n][4] = states;
            table_NFA[states - 1][4] = states;
            fin[b - 1] = states;
            table_NFA[states][0] = states;
            states += 1;
            break;
    }
  }
  return states;
}

```

```
283  void print_NFA_table(int table_NFA[][COL], int states) {
284      cout << endl;
285      for(int i = 0; i < 60; i++)
286          cout << "*";
287      cout << endl << endl;
288      cout << setw(43) << "TRANSITION TABLE FOR NFA" << endl << endl;
289      cout << setw(10) << "States" << setw(10) << "a" << setw(10) << "b" << setw(10) << "e" << setw(10) << "e" << 
290      for(int i = 0; i < 60; i++)
291          cout << "-";
292      cout << endl;
293      for(int i = 1; i < states; i++) {
294          for(int j = 0; j < COL; j++) {
295              if(table_NFA[i][j] == -1)
296                  cout << setw(10) << "--";
297              else
298                  cout << setw(10) << table_NFA[i][j];
299          }
300          cout << endl;
301      }
302      cout << endl;
303      for(int i = 0; i < 60; i++)
304          cout << "*";
305      cout << endl;
306      print_initial_final();
307  }
308
309  void print_DFA_table(string table_DFA[][3], int state) {
310      cout << endl << endl;
311      for(int i = 0; i < 60; i++)
312          cout << "*";
313      cout << endl << endl;
314      cout << setw(43) << "TRANSITION TABLE FOR DFA" << endl << endl;
315      cout << setw(10) << "States" << setw(10) << "a" << setw(10) << "b" << endl;
316      for(int i = 0; i < 60; i++)
317          cout << "-";
318      cout << endl;
319      for(int i = 0; i < state; i++){
320          for(int j = 0; j < 3; j++)
321              cout << setw(10) << table_DFA[i][j];
322          cout << endl;
323      }
324      cout << endl;
325      for(int i = 0; i < 60; i++)
326          cout << "*";
327      cout << endl;
328      print_initial_final_dfa();
329  }
330
331  vector<int> e_closure(int table_NFA[][COL], int x) {
332      stack<int> s;
333      map<int, int> m;
334      vector<int> v;
335      int y;
336      s.push(x);
337      m[x] = 1;
338      while(!s.empty()) {
339          y = s.top();
```

```
            s.pop();
            if(table_NFA[y][3] == -1)
                continue;
            else {
                s.push(table_NFA[y][3]);
                m[table_NFA[y][3]] = 1;
                if (table_NFA[y][4] == -1)
                    continue;
                else {
                    s.push(table_NFA[y][4]);
                    m[table_NFA[y][4]] = -1;
                }
            }
        }
        map<int, int>::iterator itr;
        itr = m.begin();
        while (itr != m.end()) {
            v.push_back(itr->first);
            itr++;
        }
        return v;
    }

    long long int map_it(vector<int> v, int y) {
        long long int x = 0, m = 1;
        while(y--)
            m *= 10;
        vector<int>::iterator it = v.begin();
        while(it != v.end()) {
            x += *it * m;
            m /= 10;
            it += 1;
        }
        return x / 10;
    }

    string state_name(int i) {
        char s = 'q';
        string p;
        p += s;
        if(i == 0) {
            p += '0';
            return p;
        }
        int a[100];
        int j = 0;
        while(i > 0) {
            int x = i % 10;
            a[j] = x;
            j += 1;
            i = i / 10;
        }
        for(int i = j - 1; i >= 0; i--) {
            int x = a[i];
            switch(x) {
                case 0:
```

```
396                 p += '0';
397                 break;
398
399             case 1:
400                 p += '1';
401                 break;
402
403             case 2:
404                 p += '2';
405                 break;
406
407             case 3:
408                 p += '3';
409                 break;
410
411             case 4:
412                 p += '4';
413                 break;
414
415             case 5:
416                 p += '5';
417                 break;
418
419             case 6:
420                 p += '6';
421                 break;
422
423             case 7:
```

```
424                 p += '7';
425                 break;
426
427             case 8:
428                 p += '8';
429                 break;
430
431             case 9:
432                 p += '9';
433                 break;
434         }
435     }
436     return p;
437 }
438
439 void init_CHECK(vector<int> v, string s) {
440     for(int i = 0; i < v.size(); i++) {
441         if(v[i] == init[0]) {
442             init_dfa[_a] = s;
443             _a += 1;
444         }
445     }
446 }
447
448 void fin_CHECK(vector<int> v, string s) {
449     for(int i = 0; i < v.size(); i++) {
450         if(v[i] == fin[0]) {
451             fin_dfa[_b] = s;
452             _b += 1;
453         }
454     }
455 }
456
457 bool valid_CHECK(string word) {
458     auto len = word.length();
459     int i = 0;
460     for(i = 0; i < len; i++) {
461         if(word[i] == 'a' || word[i] == 'b')
462             continue;
463         else
464             return false;
465     }
466     if(i == len)
467         return true;
468     return false;
469 }
470
471 int NFA_to_DFA(int table_NFA[][COL], int states, string table_DFA[][3]) {
472     bool flag[states];
473     memset(flag, true, sizeof(flag));
474     int state = 0, j = 0;
475     map<vector<int>, string> m_STATE;
476     vector<int> v, v1, v2, v3, v4;
477     v = e_closure(table_NFA, init[0]);
478     flag[init[a]] = false;
479     m_STATE[v] = state_name(j++);
```

```
480     init_CHECK(v, m_STATE[v]);
481     fin_CHECK(v, m_STATE[v]);
482     stack<vector<int> > st;
483     st.push(v);
484     int count = 0;
485     while(true) {
486         while(!st.empty()) {
487             vector<int> v;
488             v = st.top();
489             st.pop();
490             count += 1;
491             table_DFA[state][0] = m_STATE[v];
492             for(int i = 0; i < v.size(); i++) {
493                 flag[v[i]] = false;
494                 int temp = table_NFA[v[i]][1];
495                 int temp1 = table_NFA[v[i]][2];
496                 if (temp >= 0)
497                     v1.push_back(temp);
498                 if (temp1 >= 0)
499                     v3.push_back(temp1);
500             }
501             map<int, int> map_temp, map_temp1;
502             map<int, int>::iterator it;
503             for(int i = 0; i < v1.size(); i++) {
504                 v2 = e_closure(table_NFA, v1[i]);
505                 for(int j = 0; j < v2.size(); j++)
506                     map_temp[v2[j]] = 1;
507                 v2.clear();
508             }
509             for(int i = 0; i < v3.size(); i++) {
510                 v4 = e_closure(table_NFA, v3[i]);
511                 for(int j = 0; j < v4.size(); j++)
512                     map_temp1[v4[j]] = 1;
513                 v4.clear();
514             }
515             for(it = map_temp.begin(); it != map_temp.end(); it++) {
516                 v2.push_back(it->first);
517                 flag[it->first] = false;
518             }
519             for(it = map_temp1.begin(); it != map_temp1.end(); it++) {
520                 v4.push_back(it->first);
521                 flag[it->first] = false;
522             }
523             if(v2.empty())
524                 table_DFA[state][1] = "--";
525             else {
526                 string t = m_STATE[v2];
527                 char flagg = t[0];
528                 if(flagg == 'q')
529                     table_DFA[state][1] = m_STATE[v2];
530                 else {
531                     table_DFA[state][1] = state_name(j++);
532                     m_STATE[v2] = table_DFA[state][1];
533                     init_CHECK(v2, m_STATE[v2]);
534                     fin_CHECK(v2, m_STATE[v2]);
535                     st.push(v2);
```

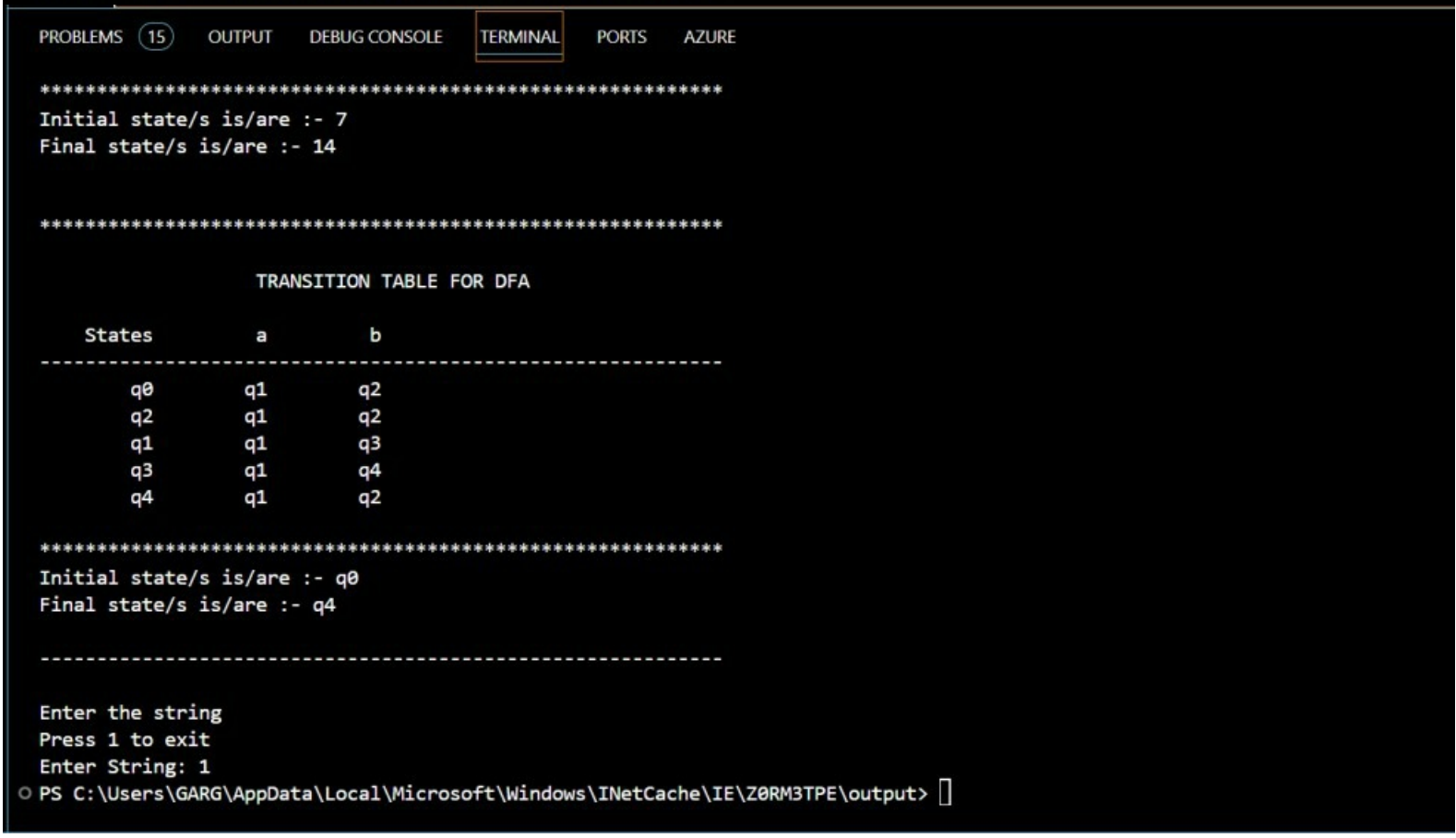
```
536                 }
537             }
538             if(v4.empty())
539                 table_DFA[state][2] = "--";
540             else {
541                 string t = m_STATE[v4];
542                 char flagg = t[0];
543                 if(flagg == 'q')
544                     table_DFA[state][2] = m_STATE[v4];
545                 else {
546                     table_DFA[state][2] = state_name(j++);
547                     m_STATE[v4] = table_DFA[state][2];
548                     init_CHECK(v4, m_STATE[v4]);
549                     fin_CHECK(v4, m_STATE[v4]);
550                     st.push(v4);
551                 }
552             }
553             v1.clear();
554             v2.clear();
555             v3.clear();
556             v4.clear();
557             state += 1;
558         }
559         int k = 1;
560         for(k = 1; k < states; k++) {
561             if(flag[k]) {
562                 v = e_closure(table_NFA, k);
563                 m_STATE[v] = state_name(j++);
```

```
564                 init_CHECK(v, m_STATE[v]);
565                 fin_CHECK(v, m_STATE[v]);
566                 cout << endl << m_STATE[v] << " represents :- ";
567                 for(int i = 0; i < v.size(); i++)
568                     cout << v[i] << " ";
569                 cout << endl;
570                 st.push(v);
571                 break;
572             }
573         }
574         if(k == states)
575             break;
576     }
577     print_DFA_table(table_DFA, state);
578     return state;
579 }
580
581 void simulator(string table_DFA[][3], string word, int state) {
582     auto len = word.length();
583     string temp = init_dfa[0];
584     bool check = valid_CHECK(word);
585     if(!check)
586         temp = "--";
587     int i = 0;
588     for(i = 0; i < len; i++) {
589         if(temp == "--") {
590             cout << endl << "String does not belong to given RE." << endl << endl << endl;
591             break;
```

```
        }
        else {
            int j = 0;
            for(j = 0; j < state; j++)
                if(temp == table_DFA[j][0])
                    break;
            if(word[i] == 'a')
                temp = table_DFA[j][1];
            else if(word[i] == 'b')
                temp = table_DFA[j][2];
        }
    }
    if(i == len) {
        int j = 0;
        for(j = 0; j < _b; j++) {
            if(temp == fin_dfa[j]) {
                cout << endl << "String belongs to given RE." << endl << endl;
                break;
            }
        }
        if(j == _b)
            cout << endl << "String does not belongs to given RE." << endl << endl;
    }
}

int main() {
    int table_NFA[1000][COL];
    initialise(table_NFA);
    int states = 0;
    cout << "Regular Expression to DFA:" << endl;
    cout << "Enter a regular expression like (a|b)*abb: ";
    string s;
    cin >> s;
    s = preprocessor(s);
    s = postfix(s);

    // Thompson's Construction
    states = re_to_nfa(s, table_NFA);
    print_NFA_table(table_NFA, states);

    // Subset Construction
    string table_DFA[1000][3]; // Adjust the size if necessary
    int State_DFA = NFA_to_DFA(table_NFA, states, table_DFA);

    while(true) {
        string word;
        cout << "Enter the string" << endl;
        cout << "Press 1 to exit" << endl;
        cout << "Enter String: ";
        cin >> word;
        if(word == "1")
            break;
        simulator(table_DFA, word, State_DFA);
    }
    return 0;
}
```

OUTPUT:

```
PROBLEMS 15  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

PS C:\Users\GARG> cd 'c:\Users\GARG\AppData\Local\Microsoft\Windows\INetCache\IE\Z0RM3TPE\output'
PS C:\Users\GARG\AppData\Local\Microsoft\Windows\INetCache\IE\Z0RM3TPE\output> & .\'Assignment-1``[1``].exe'
Regular Expression to DFA:
Enter a regular expression like (a|b)*abb: (a|b)*abb

**********************************************************

                  TRANSITION TABLE FOR NFA

    States         a         b         e         e
----------------------------------------------------------
         1         2        --        --        --
         2        --        --         6        --
         3        --         4        --        --
         4        --        --         6        --
         5        --        --         3         1
         6        --        --         5         8
         7        --        --         5         8
         8        --        --         9        --
         9        10        --        --        --
        10        --        --        11        --
        11        --        12        --        --
        12        --        --        13        --
        13        --        14        --        --
        14        --        --        --        --

**********************************************************
```



```
PROBLEMS 15  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

**********************************************************
Initial state/s is/are :- 7
Final state/s is/are :- 14


**********************************************************

                  TRANSITION TABLE FOR DFA

    States         a         b
----------------------------------------------------------
        q0        q1        q2
        q2        q1        q2
        q1        q1        q3
        q3        q1        q4
        q4        q1        q2

**********************************************************
Initial state/s is/are :- q0
Final state/s is/are :- q4

----------------------------------------------------------

Enter the string
Press 1 to exit
Enter String: 1
PS C:\Users\GARG\AppData\Local\Microsoft\Windows\INetCache\IE\Z0RM3TPE\output>
```

Name: Saneha Garg Roll No.: 102117168

Batch: 4CS6