Institute of Technology of Cambodia

Department Information Technology and Communication

Topic : **Develop an application to manage and manipulate finite automata**

Professor : Dr. **Valy Dona**

Mr. BOU Channa

Mrs. Chhou Vanna

Group: 7

PROJECT TECHNICAL REPORT SAMPLE

NAME	**ID**
Chhay Lyan	e20180087
Kheng Piseth	e20180399
Em Hengly	e20180226
Yorn Vanda	e20181287
Somoeurn Virakden	e20181001

2020~2021

Table of Contents

Introduction

This is the first project that combined 3 subjects together. Algorithm Programming, Auto Theory and Data Base and we used C ++ language in CodeBlocks for our process. Particularly, the variables, we need to discuss many times about this and it's so hard and struggle to finish task faster. However, we still finish with tremendous achievement. We learn many things more than lessons we have been study, which is like practicing the knowledge we have learned at School. In addition, this project allows us to solve problems in Auto Theory Subject for calculate finite automata and construct an equivalent DFA from an NFA easily and save previous data securely. Before the final release we had already fixed various bugs that occurred when testing and some advice from our Lecturers **Bou Channa, Valy Dona** and **Chhou Vanna** who gavesome feedback to us.

Functions and Features

a. **Design a finite automaton (FA)**
   In this function it is first screen where user need input data.

b. **Test if a FA is deterministic or non-deterministic**
   We Can know it is DFA or NFA when user input any state we can see transition.

c. **Test if a string is accepted by a FA**
   System can test string when user input any string, it display accept if it is sub-string or not it is not sub-string.

d. **Construct an equivalent DFA from an NFA.**
   In this function if user check it is NFA and then want to convert to DFA is can convert in this function.

e. **Minimize a DFA**
   If have state unused and overlapping in this function can deleted state.

Data Structure

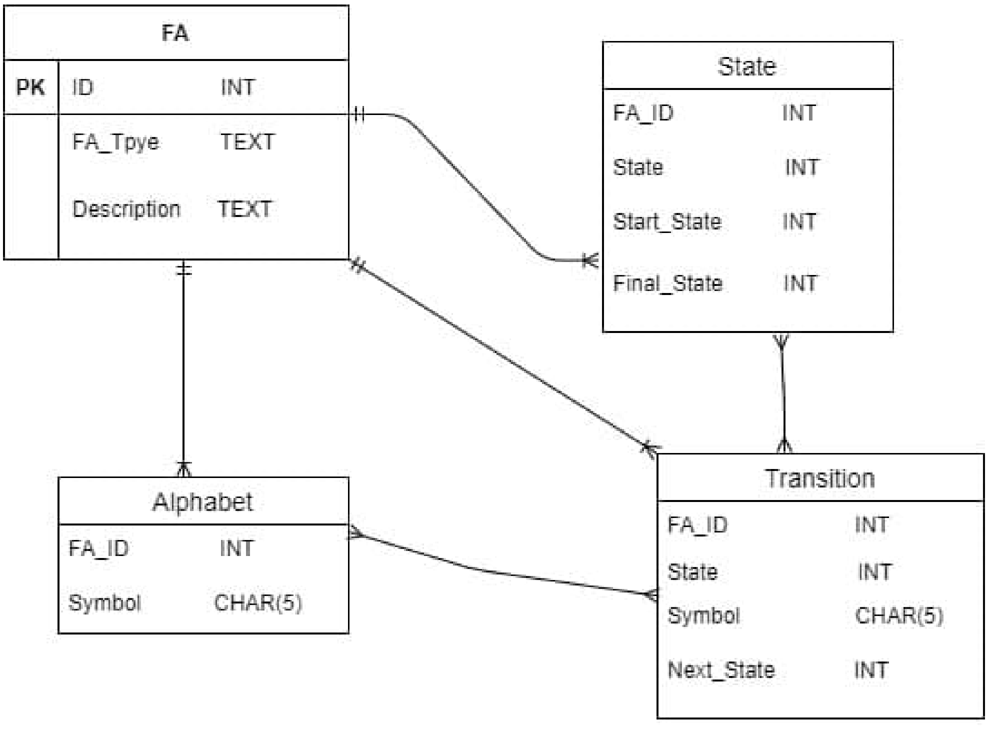
In our project use three structure has:

1. **Queue:** We use it for store transition and then it has space dynamic easy to need a lot of request and profit space.
2. **Array:** Use for store row from database and easy user select Manu.
3. **Graph:** is like Link list structure we use it for store transition.

Database Design

We make four table in our project has FA table, Alphabet table, State table and Transition table.

1. Table FA has three attributes that ID is a Primary Key type integer, FA_Type type text and Description type integer. Table FA has relationship with Alphabet table, State table and Transition table.
2. In State table store FA_ID, number State, number start state and final state.
3. In table Alphabet it has two attributes FA_ID and Symbol.
4. Table Transition have four attributes FA_ID, State, symbol and Next_State it has relationship with Table Fa, Table State and Table Alphabet.



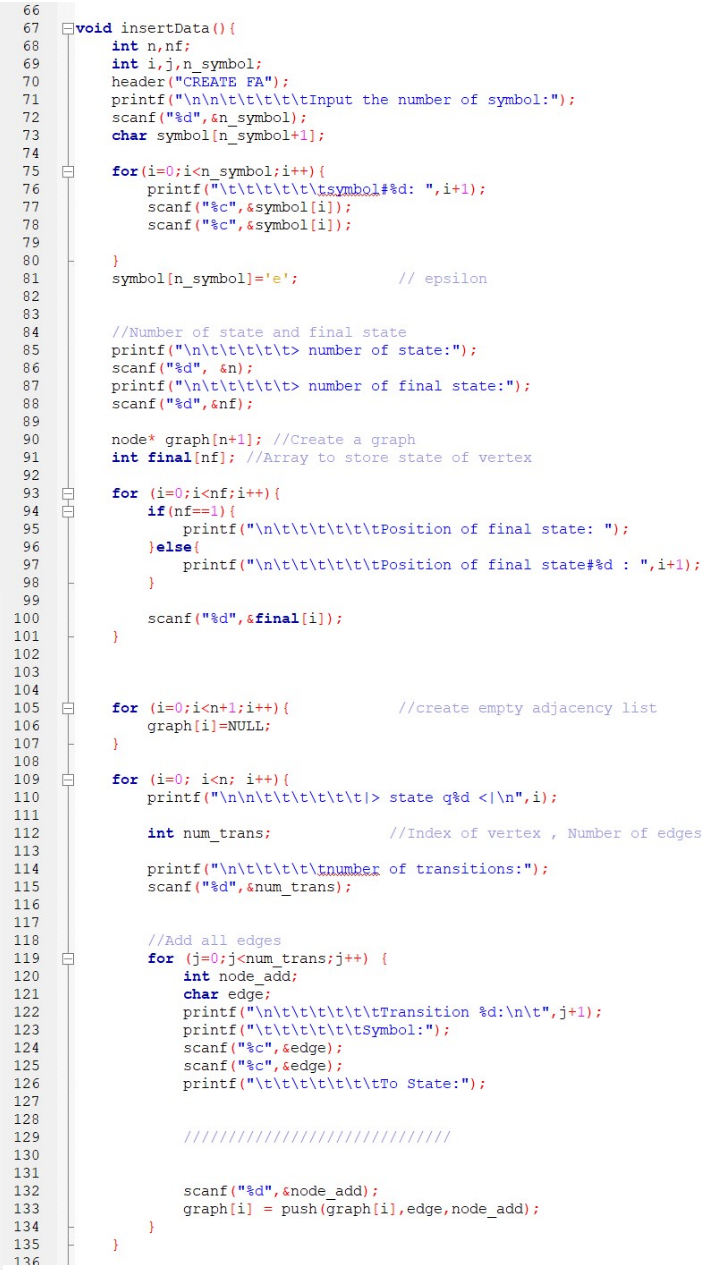
Implementation

1. Design FA:

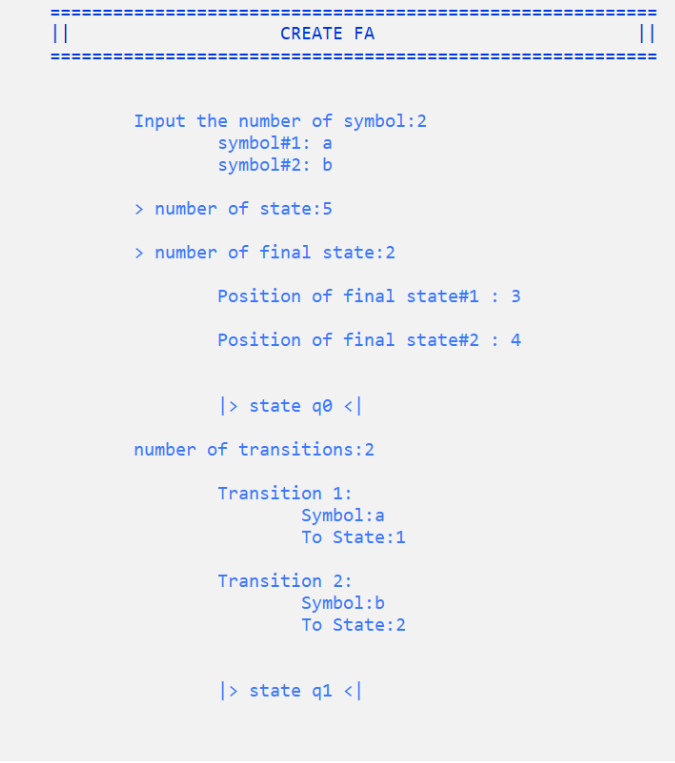
- User graph for store transition from user input.

- In function need user input:

  + Number of symbol
  + Number of state
  + Number of final state
  + Number of transition



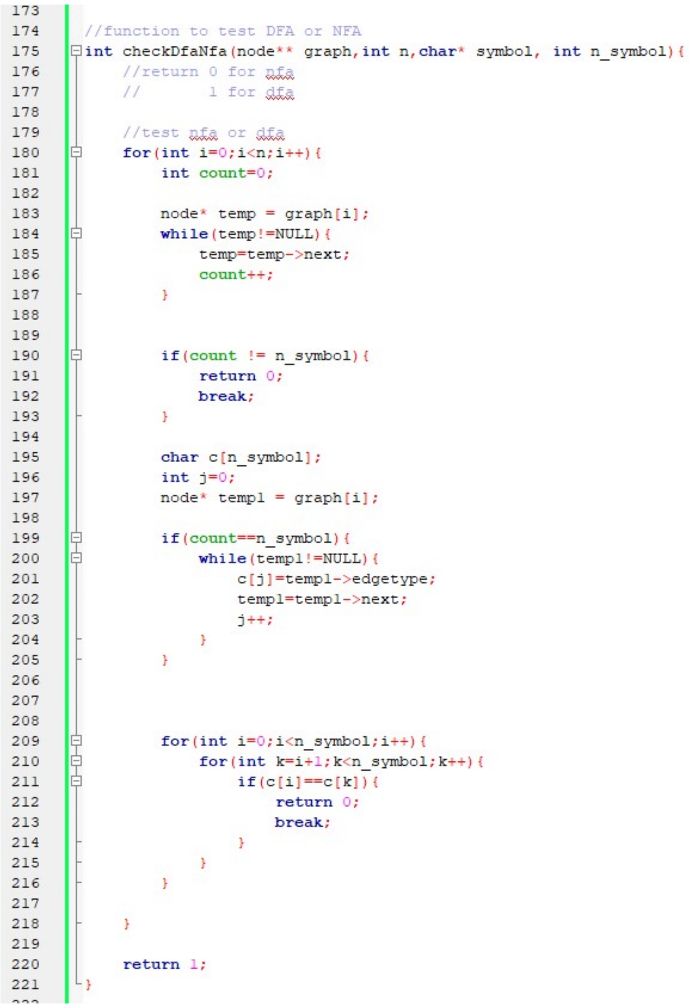
```
void insertData(){
    int n,nf;
    int i,j,n_symbol;
    header("CREATE FA");
    printf("\n\n\t\t\t\t\tInput the number of symbol:");
    scanf("%d",&n_symbol);
    char symbol[n_symbol+1];

    for(i=0;i<n_symbol;i++){
        printf("\t\t\t\t\t\tsymbol#%d: ",i+1);
        scanf("%c",&symbol[i]);
        scanf("%c",&symbol[i]);

    }
    symbol[n_symbol]='e';           // epsilon


    //Number of state and final state
    printf("\n\t\t\t\t\t> number of state:");
    scanf("%d", &n);
    printf("\n\t\t\t\t\t> number of final state:");
    scanf("%d",&nf);

    node* graph[n+1]; //Create a graph
    int final[nf]; //Array to store state of vertex

    for (i=0;i<nf;i++){
        if(nf==1){
            printf("\n\t\t\t\t\t\tPosition of final state: ");
        }else{
            printf("\n\t\t\t\t\t\tPosition of final state#%d : ",i+1);
        }

        scanf("%d",&final[i]);
    }



    for (i=0;i<n+1;i++){            //create empty adjacency list
        graph[i]=NULL;
    }

    for (i=0; i<n; i++){
        printf("\n\n\t\t\t\t\t\t|> state q%d <|\n",i);

        int num_trans;              //Index of vertex , Number of edges

        printf("\n\t\t\t\t\tnumber of transitions:");
        scanf("%d",&num_trans);


        //Add all edges
        for (j=0;j<num_trans;j++) {
            int node_add;
            char edge;
            printf("\n\t\t\t\t\t\tTransition %d:\n\t",j+1);
            printf("\t\t\t\t\t\tSymbol:");
            scanf("%c",&edge);
            scanf("%c",&edge);
            printf("\t\t\t\t\t\t\tTo State:");


            //////////////////////////////


            scanf("%d",&node_add);
            graph[i] = push(graph[i],edge,node_add);
        }
    }
```



```
==========================================================
||                    CREATE FA                         ||
==========================================================


        Input the number of symbol:2
                symbol#1: a
                symbol#2: b

        > number of state:5

        > number of final state:2

                Position of final state#1 : 3

                Position of final state#2 : 4


                |> state q0 <|

        number of transitions:2

                Transition 1:
                        Symbol:a
                        To State:1

                Transition 2:
                        Symbol:b
                        To State:2


                |> state q1 <|
```

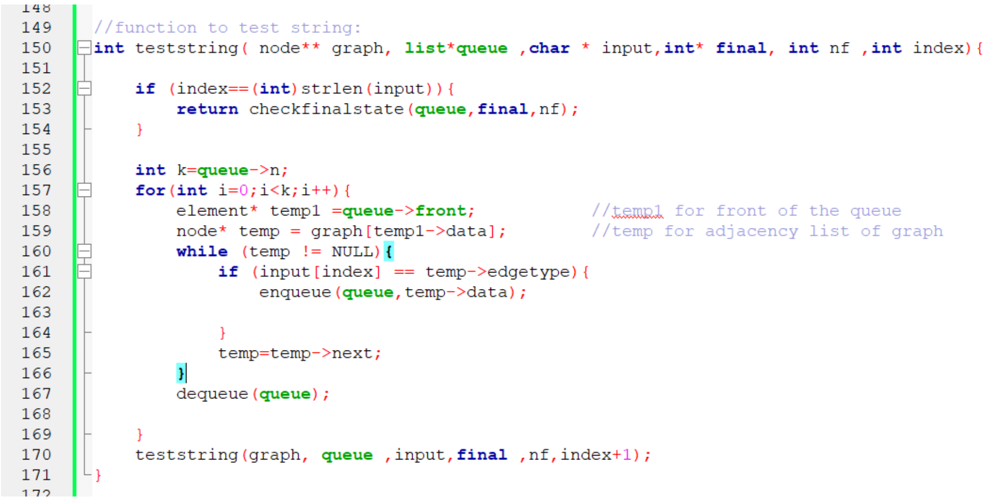
2. Test FA is deterministic or non-deterministic

In this function we use Loop and pointer for check DFA and NFA. You can see the code at the right side if the number of transitions is less or more than the number of symbol it will return o that means it is NFA. Otherwise, if the number of transition and number of symbol is equal , then we check whether each transition use different symbol. if yes, then it will return 1 and it is DFA.



```
//function to test DFA or NFA
int checkDfaNfa(node** graph,int n,char* symbol, int n_symbol){
    //return 0 for nfa
    //       1 for dfa

    //test nfa or dfa
    for(int i=0;i<n;i++){
        int count=0;

        node* temp = graph[i];
        while(temp!=NULL){
            temp=temp->next;
            count++;
        }


        if(count != n_symbol){
            return 0;
            break;
        }

        char c[n_symbol];
        int j=0;
        node* templ = graph[i];

        if(count==n_symbol){
            while(templ!=NULL){
                c[j]=templ->edgetype;
                templ=templ->next;
                j++;
            }
        }



        for(int i=0;i<n_symbol;i++){
            for(int k=i+1;k<n_symbol;k++){
                if(c[i]==c[k]){
                    return 0;
                    break;
                }
            }
        }

    }

    return 1;
}
```

3. Test string

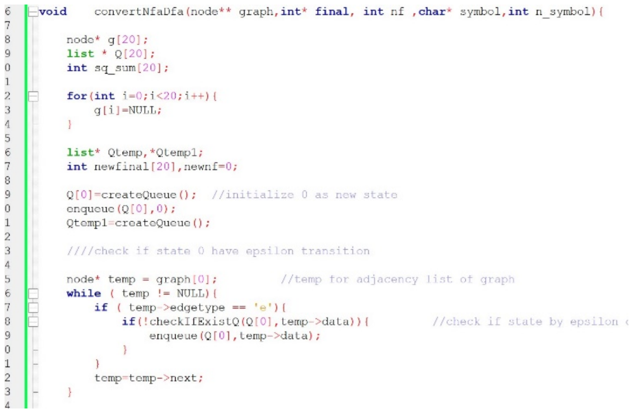


```
//function to test string:
int teststring( node** graph, list*queue ,char * input,int* final, int nf ,int index){

    if (index==(int)strlen(input)){
        return checkfinalstate(queue,final,nf);
    }

    int k=queue->n;
    for(int i=0;i<k;i++){
        element* temp1 =queue->front;           //temp1 for front of the queue
        node* temp = graph[temp1->data];        //temp for adjacency list of graph
        while (temp != NULL){
            if (input[index] == temp->edgetype){
                enqueue(queue,temp->data);

            }
            temp=temp->next;
        }
        dequeue(queue);

    }
    teststring(graph, queue ,input,final ,nf,index+1);
}
```

- Put Graph structure to store transition.
- Using recurvsive to transition by each degit of input and store in Queue of all possible state .
- Then we check whether there is final in Queue.
- If exist, then string is accepted.

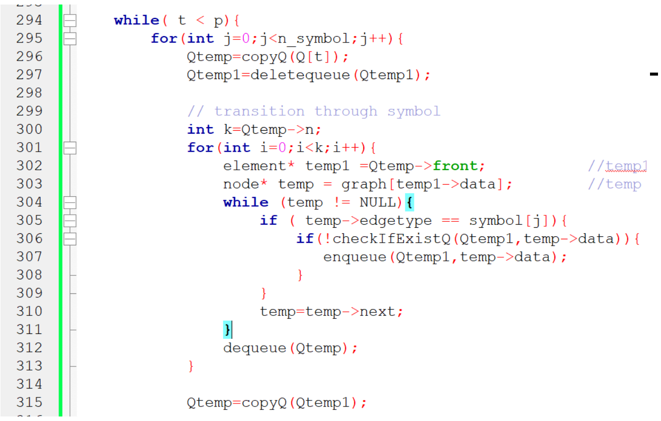
4. Convert DFA from a NFA

- Check if state 0 have epsilon transition we use queue



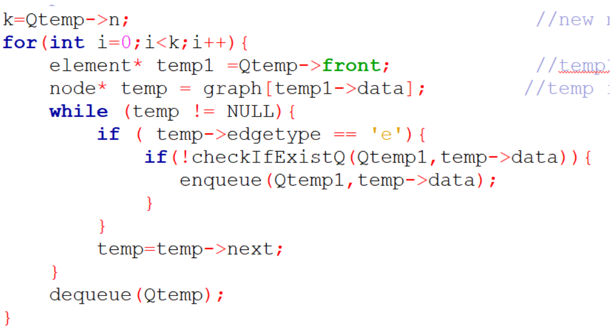
```
void    convertNfaDfa(node** graph,int* final, int nf ,char* symbol,int n_symbol){

    node* g[20];
    list * Q[20];
    int sq_sum[20];

    for(int i=0;i<20;i++){
        g[i]=NULL;
    }

    list* Qtemp,*Qtemp1;
    int newfinal[20],newnf=0;

    Q[0]=createQueue();  //initialize 0 as new state
    enqueue(Q[0],0);
    Qtemp1=createQueue();

    ////check if state 0 have epsilon transition

    node* temp = graph[0];          //temp for adjacency list of graph
    while ( temp != NULL){
        if ( temp->edgetype == 'e'){
            if(!checkIfExistQ(Q[0],temp->data)){          //check if state by epsilon
                enqueue(Q[0],temp->data);
            }
        }
        temp=temp->next;
    }
```

- We do transition by symbol.



```
    while( t < p){
        for(int j=0;j<n_symbol;j++){
            Qtemp=copyQ(Q[t]);
            Qtemp1=deletequeue(Qtemp1);

            // transition through symbol
            int k=Qtemp->n;
            for(int i=0;i<k;i++){
                element* temp1 =Qtemp->front;           //temp1
                node* temp = graph[temp1->data];        //temp
                while (temp != NULL){
                    if ( temp->edgetype == symbol[j]){
                        if(!checkIfExistQ(Qtemp1,temp->data)){
                            enqueue(Qtemp1,temp->data);
                        }
                    }
                    temp=temp->next;
                }
                dequeue(Qtemp);
            }

            Qtemp=copyQ(Qtemp1);
```

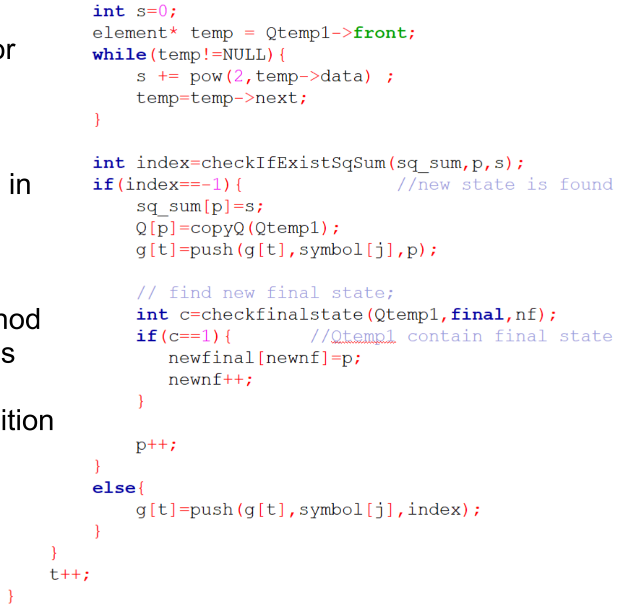
- We do transition by epsilon.



```
k=Qtemp->n;                                          //new
for(int i=0;i<k;i++){
    element* temp1 =Qtemp->front;                    //temp
    node* temp = graph[temp1->data];                 //temp
    while (temp != NULL){
        if ( temp->edgetype == 'e'){
            if(!checkIfExistQ(Qtemp1,temp->data)){
                enqueue(Qtemp1,temp->data);
            }
        }
        temp=temp->next;
    }
    dequeue(Qtemp);
}
```

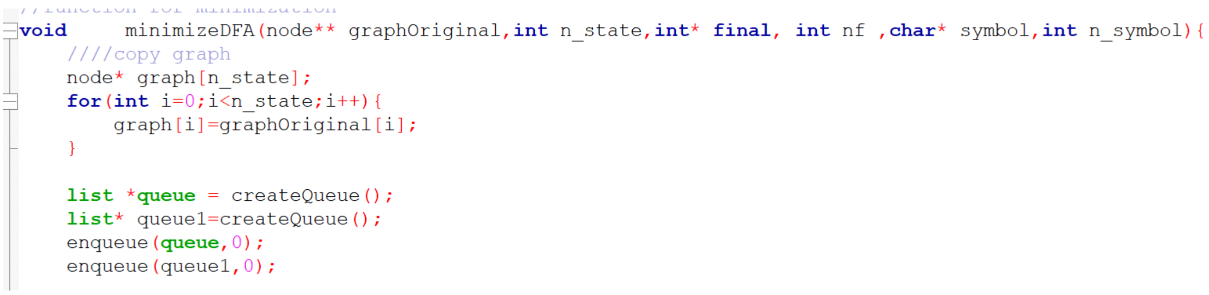
- In this function we use formula for solution is $2^0 + 2^1 + ---- + 2^n$ (n = transition store in queue)

- In formula call sq_sum and store in array.
- And then check sq_sum already accept or not.
- Finally we get states by this method ,we can get new number of states that is equivalent DFA
- and g[t] is a graph for store transition of equivalent DFA



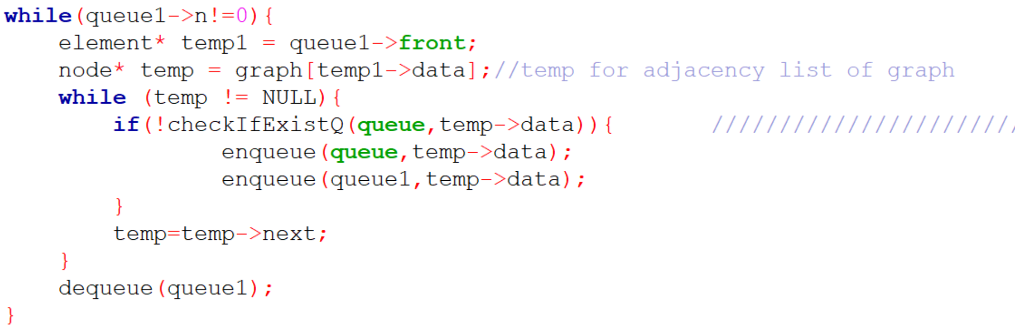
```
        int s=0;
        element* temp = Qtemp1->front;
        while(temp!=NULL){
            s += pow(2,temp->data) ;
            temp=temp->next;
        }

        int index=checkIfExistSqSum(sq_sum,p,s);
        if(index==-1){              //new state is found
            sq_sum[p]=s;
            Q[p]=copyQ(Qtemp1);
            g[t]=push(g[t],symbol[j],p);

            // find new final state;
            int c=checkfinalstate(Qtemp1,final,nf);
            if(c==1){        //Qtemp1 contain final state
               newfinal[newnf]=p;
               newnf++;
            }

            p++;
        }
        else{
            g[t]=push(g[t],symbol[j],index);
        }
    }
    t++;
}
```

5. Minimize a DFA



```
//function for minimization
void     minimizeDFA(node** graphOriginal,int n_state,int* final, int nf ,char* symbol,int n_symbol){
    ////copy graph
    node* graph[n_state];
    for(int i=0;i<n_state;i++){
        graph[i]=graphOriginal[i];
    }

    list *queue = createQueue();
    list* queue1=createQueue();
    enqueue(queue,0);
    enqueue(queue1,0);
```

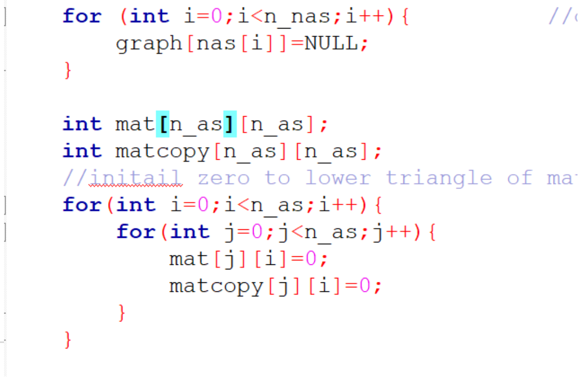
- In this function have graph and queue structure, n_state, final, nf, symbol and n-symbol.
- Next copy graph structure put in this function.



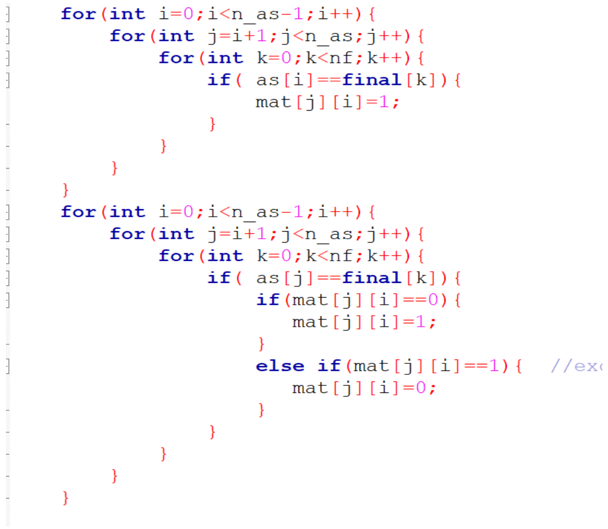
```
while(queue1->n!=0){
    element* temp1 = queue1->front;
    node* temp = graph[temp1->data];//temp for adjacency list of graph
    while (temp != NULL){
        if(!checkIfExistQ(queue,temp->data)){       //////////////////////,
                enqueue(queue,temp->data);
                enqueue(queue1,temp->data);
        }
        temp=temp->next;
    }
    dequeue(queue1);
}
```

- Use queue in while loop to find accessible state.

- Next subtract non-accessible.



```
for (int i=0;i<n_nas;i++){          //
    graph[nas[i]]=NULL;
}

int mat[n_as][n_as];
int matcopy[n_as][n_as];
//initail zero to lower triangle of ma
for(int i=0;i<n_as;i++){
    for(int j=0;j<n_as;j++){
        mat[j][i]=0;
        matcopy[j][i]=0;
    }
}
```



```
for(int i=0;i<n_as-1;i++){
    for(int j=i+1;j<n_as;j++){
        for(int k=0;k<nf;k++){
            if( as[i]==final[k]){
                mat[j][i]=1;
            }
        }
    }
}
for(int i=0;i<n_as-1;i++){
    for(int j=i+1;j<n_as;j++){
        for(int k=0;k<nf;k++){
            if( as[j]==final[k]){
                if(mat[j][i]==0){
                   mat[j][i]=1;
                }
                else if(mat[j][i]==1){   //ex
                   mat[j][i]=0;
                }
            }
        }
    }
}
```

- Use 2D array with solution matrix.
- Use loop do it all pairs.
- mark all pairs that contain final state, but not include pair of 2 final states.
- And then we delete equivalent states and re arrange of the remaining state by in order to get minimize DFA.

Result

The result is stunning for our team because we have made a brand new console application that we have never ever done before. Over 98% of what we finished and it works. It is the result that we wanted and expected. The main result we get from this project are:

- Improving and development.
- Teamwork.
- Solving problem skill.

Conclusion and Perspective

As a team, we all have faced many problems, especially the project that is newly-experienced for us. These are not the obstacles to gain the well-done result, but also can be the most significant change that we learn at school -- practical skill. Plus, in spite of a few issues still occurring, our program is run well. The project about develops an application to manage and manipulate finite automata can help us to get brand new ideas and concepts on how to solve the problem and we will reach a new one in the future by strengthening the knowledge and practical skill too. We hope for a new project from the teacher because the more project the more research, knowledge and teamwork skill.