4108056051 資工三 鄭穎

1. 程式碼：
   (1) Main function:

```cpp
int main(){
    string represent;
    discribe_rules();
    if(input_circuit()){
        draw_circuit();
        generate_qState();
        process_qubit();
        cout << "\n\nPlease choose the truth table representation\n(bin/oct/hex/dec): ";
        cin >> represent;
        cout << "\n";
        if(!represent.compare("bin"))
            bin_output();
        else if(!represent.compare("oct"))
            oct_output();
        else if(!represent.compare("hex"))
            hex_output();
        else if(!represent.compare("dec"))
            dec_output();
        else
            cout << "Invalid input of representation\n";
    }
    else
        cout << "Invalid input of circuit\n";
    return 0;
}
```

   (2) Describe_rules(): describe the input rules

```cpp
void discribe_rules(){
    cout << "The input rules and descriptions of input numbers are listed below:\n\n";
    cout << "Input\tDescription\n";
    cout << "0\tControl bit(low level trigger)\n";
    cout << "1\tControl bit(high level trigger)\n";
    cout << "2\tNOT-gate\n";
    cout << "3\tWire gate\n\n";
    cout << "Input rules: \n";
    cout << "1. Each row represents a gate\n";
    cout << "2. The number of columns represents the number of input Qubits\n";
    cout << "3. It's invalid that a gate only has control bits and wire gate and without NOT-gate\n";
    cout << "4. Please seperate each input with a blank\n";
    cout << "5. After the end of each row, please add a newline\n\n";
}
```

   (3) Input_circuit(): let user to input circuit and will return whether the gate that user input is legal

```
56  bool input_circuit(){
57      cout << "Input the number of gate: ";
58      cin >> g_num;
59      cout << "Input the number of Qubit: ";
60      cin >> q_num;
61      cout << "Input the circuit: \n";
62      char c;
63      c = getchar();//read \n
64      circuit=new char*[g_num];
65      bool valid=true;
66      int tmp;
67      for (int i = 0; i < g_num;i++){//judge if it's valid input
68          circuit[i] = new char[q_num];
69          tmp = 0;
70          for (int j = 0; j < q_num;j++){
71              c = getchar();
72              if(c==' '||c=='\n'||(c-'0')<0||(c-'0')>3){
73                  valid = false;
74                  break;
75              }
76              else{
77                  circuit[i][j] = c;
78                  if(c=='2')
79                      tmp++;
80              }
81              c = getchar();
82              if((j==q_num-1&&c=='\n')||(j!=q_num-1&&c==' '))
83                  valid = true;
84              else {
85                  valid = false;
86                  break;
87              }
88          }
```

```
89          if(!tmp)
90              valid = false;
91          if(!valid)
92              break;
93      }
94      if(!valid)
95          return false;
96      else
97          return true;
98  }
```

(4) Draw_circuit():

```
255  void draw_circuit(){
256      cout << "\nCircuit diagram\n(@: low-level control bit)\n(O: high-level control bit)\n(N: not-gate)\n";
257      for (int i = 0; i < q_num;i++){
258          cout << "q" << i;
259          for (int j = 0; j < g_num;j++){
260              switch(circuit[j][i]){
261                  case '0':
262                      cout << "--O-";
263                      break;
264                  case '1':
265                      cout << "--@-";
266                      break;
267                  case '2':
268                      cout << "--N-";
269                      break;
270                  case '3':
271                      cout << "----";
272                      break;
273              }
274              if(j==g_num-1&&i!=q_num-1){
275                  cout << "\n";
276                  for (int k = 0; k < g_num;k++){
277                      if(k==0)
278                          cout << "    | ";
279                      else
280                          cout << "  | ";
281                  }
282                  cout << "\n";
283              }
284          }
285      }
286  }
```

(5) Generate_qState(): generate every possible qubit set based on the number of given qubit numbers

```
 99  void generate_qState(){
100      int tmp;
101      for (int i = 0; i < q_num;i++)
102          qState_num *= 2;
103      origin_qubit = new char *[qState_num];
104      qubit = new char *[qState_num];
105      for (int i = 0; i < qState_num;i++){
106          origin_qubit[i] = new char[q_num];
107          qubit[i] = new char[q_num];
108          tmp = i;
109          for (int j = q_num - 1; j >= 0; j--){
110              origin_qubit[i][j] = tmp % 2 + '0';
111              qubit[i][j] = origin_qubit[i][j];
112              tmp /= 2;
113          }
114      }
115  }
```

(6) Process_qubit(): let each qubit set go through the circuit and record the result

```
116  void process_qubit(){
117      bool chg;
118      for (int i = 0; i < qState_num;i++){
119          for (int j = 0; j < g_num; j++){
120              chg = true;
121              for (int k = 0; k < q_num; k++){
122                  if((circuit[j][k]=='0'&&qubit[i][k]=='1')||(circuit[j][k]=='1'&&qubit[i][k]=='0')){
123                      chg = false;
124                      break;
125                  }
126              }
127              if(chg){
128                  for (int k = 0; k < q_num; k++){
129                      if(circuit[j][k]=='2'){
130                          if(qubit[i][k]=='0')
131                              qubit[i][k] = '1';
132                          else
133                              qubit[i][k] = '0';
134                      }
135                  }
136              }
137          }
138      }
139  }
```

(7) Choose which representation that will be used in representing truth table

i. Binary:

```
140  void bin_output(){
141      cout << "Truth table in binary representation:\n";
142      for (int i = 0; i < qState_num;i++){
143          for (int j = 0; j < q_num;j++)
144              cout << origin_qubit[i][j];
145          cout << " -> ";
146          for (int j = 0; j < q_num;j++)
147              cout << qubit[i][j];
148          cout << "\n";
149      }
150  }
```

ii. Decimal:

```
151  void dec_output(){
152      cout << "Truth table in decimal representation:\n";
153      int tmp,itr;
154      for (int i = 0; i < qState_num;i++){
155          tmp = 0;
156          itr = 1;
157          for (int j = q_num - 1; j >=0; j--){
158              tmp+=itr*(origin_qubit[i][j]-'0');
159              itr *= 2;
160          }
161          cout << tmp << " -> ";
162          tmp = 0;
163          itr = 1;
164          for (int j = q_num - 1; j >=0; j--){
165              tmp+=itr*(qubit[i][j]-'0');
166              itr *= 2;
167          }
168          cout << tmp << "\n";
169      }
170  }
```

iii. Octal:

```
171  void oct_output(){
172      cout << "Truth table in octal representation:\n";
173      int rest = q_num % 3, itr, tmp;
174      for (int i = 0; i < qState_num; i++){
175          itr = 1;
176          tmp = 0;
177          for (int j = rest-1; j>=0;j--){
178              tmp += (origin_qubit[i][j]-'0') * itr;
179              itr *= 2;
180          }
181          if(tmp)
182              cout << tmp;
183          for (int j = rest; j < q_num; j+=3){
184              tmp = 0;
185              tmp += (origin_qubit[i][j]-'0') * 4;
186              tmp += (origin_qubit[i][j+1]-'0') * 2;
187              tmp += (origin_qubit[i][j+2]-'0') * 1;
188              cout << tmp;
189          }
190          cout << " -> ";
191          itr = 1;
192          tmp = 0;
193          for (int j = rest-1; j>=0;j--){
194              tmp += (qubit[i][j]-'0') * itr;
195              itr *= 2;
196          }
197          if(tmp)
198              cout << tmp;
199          for (int j = rest; j < q_num; j+=3){
200              tmp = 0;
201              tmp += (qubit[i][j]-'0') * 4;
202              tmp += (qubit[i][j+1]-'0') * 2;
203              tmp += (qubit[i][j+2]-'0') * 1;
204              cout << tmp;
205          }
206          cout << "\n";
207      }
208  }
```

iv. Hexadecimal:

```
void hex_output(){
    cout << "Truth table in hexidecimal representation:\n";
    int rest = q_num % 4, itr, tmp;
    for (int i = 0; i < qState_num; i++){
        itr = 1;
        tmp = 0;
        for (int j = rest-1; j>=0;j--){
            tmp += (origin_qubit[i][j]-'0') * itr;
            itr *= 2;
        }
        if(tmp)
            cout << tmp;
        for (int j = rest; j < q_num; j+=4){
            tmp = 0;
            tmp += (origin_qubit[i][j]-'0') * 8;
            tmp += (origin_qubit[i][j+1]-'0') * 4;
            tmp += (origin_qubit[i][j+2]-'0') * 2;
            tmp += (origin_qubit[i][j+3]-'0') * 1;
            if(tmp>=10)
                cout << char(tmp+55);
            else
                cout << tmp;
        }
        cout << " -> ";
        itr = 1;
        tmp = 0;
        for (int j = rest-1; j>=0;j--){
            tmp += (qubit[i][j]-'0') * itr;
            itr *= 2;
        }
        if(tmp)
            cout << tmp;
        for (int j = rest; j < q_num; j+=4){
            tmp = 0;
            tmp += (qubit[i][j]-'0') * 8;
            tmp += (qubit[i][j+1]-'0') * 4;
            tmp += (qubit[i][j+2]-'0') * 2;
            tmp += (qubit[i][j+3]-'0') * 1;
            if(tmp>=10)
                cout << char(tmp+55);
            else
                cout << tmp;
        }
        cout << "\n";
    }
}
```

2. 輸出結果：

(1) Describe rules:

```
The input rules and descriptions of input numbers are listed below:

Input   Description
0       Control bit(low level trigger)
1       Control bit(high level trigger)
2       NOT-gate
3       Wire gate

Input rules:
1. Each row represents a gate
2. The number of columns represents the number of input Qubits
3. It's invalid that a gate only has control bits and wire gate and without NOT-gate
4. Please seperate each input with a blank
5. After the end of each row, please add a newline
```

(2) Input:

```
Input the number of gate: 6
Input the number of Qubit: 5
Input the circuit:
1 1 2 3 1
2 1 3 0 1
1 2 3 0 1
0 0 2 0 1
2 0 0 1 1
0 1 1 2 0
```

(3) Draw circuit:

```
Circuit diagram
(@: low-level control bit)
(O: high-level control bit)
(N: not-gate)
q0--@---N---@---O---N---O-
    |   |   |   |   |   |
q1--@---@---N---O---O---@-
    |   |   |   |   |   |
q2--N-----------N---O---@-
    |   |   |   |   |   |
q3------O---O---O---@---N-
    |   |   |   |   |   |
q4--@---@---@---@---@---O-
```

(4) Generate truth table using the representation that user choose

```
Please choose the truth table representation
(bin/oct/hex/dec): bin

Truth table in binary representation:
00000 -> 00000
00001 -> 00101
00010 -> 00010
00011 -> 10011
00100 -> 00100
00101 -> 00001
00110 -> 00110
00111 -> 00111
01000 -> 01000
01001 -> 10001
01010 -> 01010
01011 -> 01011
01100 -> 01110
01101 -> 10101
01110 -> 01100
01111 -> 01111
10000 -> 10000
10001 -> 11001
10010 -> 10010
10011 -> 00011
10100 -> 10100
10101 -> 11101
10110 -> 10110
10111 -> 10111
11000 -> 11000
11001 -> 01101
11010 -> 11010
11011 -> 11111
11100 -> 11100
11101 -> 01001
11110 -> 11110
11111 -> 11011
```