

```

1  #include "Gate.h"
2
3  // Function prototype definitions to update each gate type
4  char andUpdate(char one, char two);
5  char orUpdate(char one, char two);
6  char nandUpdate(char one, char two);
7  char norUpdate(char one, char two);
8  char xorUpdate(char one, char two);
9  char xnorUpdate(char one, char two);
10 char notUpdate(char one, char two);
11
12 // Gate constructor
13 Gate::Gate(Wire* in1, Wire* in2, Wire* out, int typ, int dly)
14 {
15     input1 = in1;
16     input2 = in2;
17     output = out;
18     type = typ;
19     delay = dly;
20 }
21
22 // Gate destructor
23 Gate::~~Gate()
24 {
25     delete input1;
26     delete input2;
27     delete output;
28 }
29
30
31 // Gate update function
32 void Gate::update(int time, priority_queue<Event>* eq, int priority)
33 {
34     // Bringing in the current wire values
35     char one = (*input1).getValue();
36     char two = (*input2).getValue();
37     char ret;
38     // Checking the gate type and calling the appropriate update function
39     switch(type)
40     {
41         case 1:
42             ret = andUpdate(one, two);
43             break;
44         case 2:
45             ret = orUpdate(one, two);
46             break;
47         case 3:
48             ret = nandUpdate(one, two);
49             break;
50         case 4:
51             ret = norUpdate(one, two);
52             break;
53         case 5:
54             ret = xorUpdate(one, two);
55             break;
56         case 6:
57             ret = xnorUpdate(one, two);
58             break;
59         case 7:
60             ret = notUpdate(one, two);
61         default:
62             break;
63     }
64
65     // Creating a new event to change the output wire
66     // after the appropriate gate delay

```

```

67     Event newEvent(time+delay, output, ret, priority);
68     (*eq).push(newEvent);
69 }
70
71 // Update functions for each type of gate
72 char andUpdate(char one, char two)
73 {
74     if((one == 'x' || two == 'x') && (one == '-' || two == '-'))
75     {
76         return 'x';
77     }
78     if(one == '-' && two == '-')
79     {
80         return '-';
81     }
82     return '_';
83 }
84
85 char orUpdate(char one, char two)
86 {
87     if((one == 'x' || two == 'x') && (one == '_' || two == '_'))
88     {
89         return 'x';
90     }
91     if(one == '-' || two == '-')
92     {
93         return '-';
94     }
95     return '_';
96 }
97
98 char nandUpdate(char one, char two)
99 {
100     if((one == 'x' || two == 'x') && (one == '-' || two == '-'))
101     {
102         return 'x';
103     }
104     if(one == '-' && two == '-')
105     {
106         return '_';
107     }
108     return '-';
109 }
110
111 char norUpdate(char one, char two)
112 {
113     if((one == 'x' || two == 'x') && (one == '_' || two == '_'))
114     {
115         return 'x';
116     }
117     if(one == '_' && two == '_')
118     {
119         return '-';
120     }
121     return '_';
122 }
123
124 char xorUpdate(char one, char two)
125 {
126     if(one == 'x' || two == 'x')
127     {
128         return 'x';
129     }
130     if(one == '-' && two == '_')
131     {
132         return '-';

```

```
133     }
134     if(one == '_' && two == '-')
135     {
136         return '-';
137     }
138     return '_';
139 }
140
141 char xnorUpdate(char one, char two)
142 {
143     if(one == 'x' || two == 'x')
144     {
145         return 'x';
146     }
147     if(one == '-' && two == '_')
148     {
149         return '_';
150     }
151     if(one == '_' && two == '-')
152     {
153         return '_';
154     }
155     return '-';
156 }
157
158 char notUpdate(char one, char two)
159 {
160     if(one == 'x')
161     {
162         return 'x';
163     }
164     if(one == '-')
165     {
166         return '_';
167     }
168     if(one == '_')
169     {
170         return '-';
171     }
172 }
```