```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <queue>

#include "Output.h"
#include "Gate.h"
#include "Wire.h"
#include "Event.h"
#include "Input.h"

using namespace std;

int main()
{
    ifstream in;

    string input;
    string crctName;
    string crctFile;
    string vectFile;
    string inLine;
    string part;
    string tag;
    string temp;

    int numcycl;
    int numtag;
    int loc;
    int loc1;
    int delay;
    int wire1;
    int wire2;
    int outwire;
    int type;
    int time;
    int priority;

    char tempc;

    vector<Wire*> wires;
    vector<Gate*> gates;
    vector<Input> inputs;
    vector<Output> outputs;

    priority_queue<Event> *eq = new priority_queue<Event>;

    // Getting circuit name
    cout<<"Welcome to SchneiderLOGIC Circuit Simulator!\n";
    cout<<"Please enter the base name of your circuit:\n";
    cout<<"~$ ";
    getline(cin, input, '\n');
    crctFile = input + ".txt";
    vectFile = input + "_vector.txt";

    // Opening circuit file
    in.open(crctFile.c_str(), ifstream::in);

    // Checking for existence
    while(!in)
    {
        cout<<"\nError: file "<<crctFile<<" does not exist.  Please try again.\n";
        cout<<"~$ ";
        getline(cin, input, '\n');
        crctFile = input + ".txt";
```

```
        vectFile = input + "_vector.txt";
        in.open(crctFile.c_str(), ifstream::in);
    }

    // Getting the first line (title line) and then taking in the first meaningful line
    getline(in, inLine, '\n');
    crctName = inLine;
    getline(in, inLine, '\n');

    // Processing all inputs and outputs
    while(inLine.substr(0,1)=="I"||(inLine.substr(0,2)=="OU"))
    {
        // Parsing the string to find the input/output tag
        loc = inLine.find(" ");
        while(inLine[loc]==' ')
        {
            loc++;
        }
        part = inLine.substr(loc, inLine.size()-loc);
        loc1 = part.find(" ");
        tag = inLine.substr(loc, loc1);

        // Parsing the string to find the wire index number and converting to int
        loc += loc1;
        while(inLine[loc]==' ')
        {
            loc++;
        }
        part = inLine.substr(loc, inLine.size()-loc1);
        numtag = atoi(part.c_str());

        // Checking to ensure that the tag is legal
        if(numtag<=0)
        {
            cout<<"Error: Wire tag less than or equal to zero."<<endl;
            return 0;
        }
        numtag++;

        // Resizing the wire vector to accommodate the tag of the new wire if necessary
        if(numtag>wires.size())
        {
            for(int i = numtag; i>(wires.size()); i--)
            {
                wires.push_back(NULL);
            }
        }
        if(numtag>wires.size())
        {
            wires.push_back(NULL);
        }
        numtag--;

        // Setting the wire in the correct vector index and increasing the vector size
        // Also creating a new input or output with the correct tag and linked to the correct wire
        if(inLine.substr(0,1)=="I")
        {
            if(wires[numtag]!=NULL)
            {
                cout<<"Error: Two inputs controlling the same wire."<<endl;
                return 0;
            }
            Wire* newWire = new Wire(numtag);
            wires[numtag] = newWire;
            Input newInput(tag, newWire);
            inputs.push_back(newInput);
```

```
        }
        if(inLine.substr(0,1)=="O")
        {
            if(wires[numtag]==NULL)
            {
                Wire* newWire = new Wire(numtag);
                wires[numtag] = newWire;
            }

            Output newOutput(tag, wires[numtag]);
            outputs.push_back(newOutput);
        }

        // Getting the next line
        getline(in, inLine, '\n');
    }

    // Parsing the file to find and load each gate
    while(!in.eof())
    {
        wire2 = 0;

        // Parsing the string to find the gate type
        loc = inLine.find(" ");
        tag = inLine.substr(0,loc);
        loc++;

        // Parsing the string to find the gate delay
        while(inLine[loc]==' ')
        {
            loc++;
        }
        part = inLine.substr(loc, inLine.size()-loc);
        loc1 = part.find("ns");
        temp = part.substr(0, loc1);
        delay = atoi(temp.c_str());
        loc = part.find(" ");
        while(part[loc]==' ')
        {
            loc++;
        }

        // Parsing the string to find first input wire
        part = part.substr(loc, part.size()-loc);
        loc = part.find(" ");
        temp = part.substr(0, loc);
        wire1 = atoi(temp.c_str());
        while(part[loc]==' ')
        {
            loc++;
        }

        if(tag!="NOT")
        {
            // Parsing the string to find second input wire
            part = part.substr(loc, part.size()-loc);
            loc = part.find(" ");
            temp = part.substr(0, loc);
            wire2 = atoi(temp.c_str());
        }

        // Parsing the string to find outwire
        loc = part.find(" ");
        while(part[loc]==' ')
        {
            loc++;
```

```
199         }
200         part = part.substr(loc, part.size()-loc);
201         outwire = atoi(part.c_str());
202 
203         // Putting the gate on the vector
204         type = 0;
205         if(tag == "AND")
206         {
207             type = 1;
208         }
209         if(tag == "OR")
210         {
211             type = 2;
212         }
213         if(tag == "NAND")
214         {
215             type = 3;
216         }
217         if(tag == "NOR")
218         {
219             type = 4;
220         }
221         if(tag == "XOR")
222         {
223             type = 5;
224         }
225         if(tag == "XNOR")
226         {
227             type = 6;
228         }
229         if(tag == "NOT")
230         {
231             type = 7;
232         }
233         if(type == 0)
234         {
235             cout<<"Error: Gate type "<<tag<<"not recognized."<<endl;
236             return 0;
237         }
238 
239         // Making a new gate and putting it on the vector
240         // Finding the largest wiretag and increasing the vector size if needed
241         if(wire1>=wire2)
242         {
243             if(wire1>=outwire)
244             {
245                 numtag = wire1;
246             }
247             else
248             {
249                 numtag = outwire;
250             }
251         }
252         else
253         {
254             if(wire2>=outwire)
255             {
256                 numtag = wire2;
257             }
258             else
259             {
260                 numtag = outwire;
261             }
262         }
263 
264         if(numtag>wires.size())
```

```
        {
            for(int i = numtag; i>(wires.size()); i--)
            {
                wires.push_back(NULL);
            }
        }
        // Creating a new wire with the right tag if needed
        if(wires[wire1]==NULL)
        {
            Wire* newWire = new Wire(wire1);
            wires[wire1] = newWire;
        }
        if(type!=7 && wires[wire2]==NULL)
        {
            Wire* newWire = new Wire(wire2);
            wires[wire2] = newWire;
        }
        if(wires[outwire]==NULL)
        {
            Wire* newWire = new Wire(outwire);
            wires[outwire] = newWire;
        }
        if(type == 7)
        {
            wire2=wire1;
        }
        Gate* newGate = new Gate(wires[wire1],wires[wire2],wires[outwire],type, delay);

        gates.push_back(newGate);

        // Adding the gate to each wire's driving vector
        (*wires[wire1]).addToDriving(newGate);
        // If statement so if the same wire drives both inputs of a gate,
        // the gate isn't added twice
        if(wires[wire1]!=wires[wire2])
        {
            (*wires[wire2]).addToDriving(newGate);
        }

        // Getting the next line
        getline(in, inLine, '\n');
        // So the program doesn't break for blank lines at the end of a file
        if(inLine == "")
        {
            break;
        }
    }

    // Closing the circuit file and opening the vector file
    in.close();
    in.open(vectFile.c_str());
    priority = 0;

    // Checking for existence
    while(!in)
    {
        cout<<"\nError: file "<<crctFile<<"_vector.txt does not exist.  Please try again.\n";
        cout<<"~$ ";
        getline(cin, input, '\n');
        vectFile = input + "_vector.txt";
        in.open(vectFile.c_str(), ifstream::in);
    }
    // Getting the number of cycles to simulate
    cout<<"\nHow many cycles would you like to simulate?\n";
    cout<<"~$ ";
    cin>>numcycl;
```

```
331
332     // Discarding the title line and taking the first meaningful line
333     getline(in, inLine, '\n');
334     getline(in, inLine, '\n');
335
336     while(!in.eof() && inLine!="")
337     {
338         // Discarding the "Input" and every space after it
339         loc = inLine.find(" ");
340         while(inLine[loc]==' ')
341         {
342             loc++;
343         }
344         // Finding the tag
345         part = inLine.substr(loc,inLine.size()-loc);
346         loc = part.find(" ");
347         tag = part.substr(0,loc);
348
349         // Finding the time
350         while(part[loc] == ' ')
351         {
352             loc++;
353         }
354         part = part.substr(loc, part.size()-loc);
355         loc = part.find(" ");
356         temp = part.substr(0,loc);
357         wire1 = atoi(temp.c_str());
358
359         // Finding the new value
360         while(part[loc]==' ')
361         {
362             loc++;
363         }
364         part = part.substr(loc, part.size()-loc);
365         tempc = part[0];
366
367         switch(tempc)
368         {
369             case '1':
370             tempc = '-';
371             break;
372             case '0':
373             tempc = '_';
374             break;
375             default:
376             cout<<"Error: All wire values must be 1 or 0.  Value "<<tempc<<" is not allowed."<<endl;
377             return 0;
378         }
379
380         // Finding the input that corresponds
381         bool test = false;
382         for(int i=0; i<inputs.size(); i++)
383         {
384             if(inputs[i].getTag()==tag)
385             {
386                 // Pushing a new event using that input's Wire
387                 Event newEv(wire1, inputs[i].getWire(), tempc, priority);
388                 (*eq).push(newEv);
389                 test = true;
390             }
391         }
392
393         // Error if a nonexistent input was modified in the vector file
394         if(!test)
395         {
396             cout<<"Error: Attempting to change nonexisting input "<<tag<<"."<<endl;
```

```
            return 0;
        }
        priority++;
        getline(in, inLine, '\n');
    }

    // Running the simulation
    for(time=0; time<numcycl; time++)
    {
        // Check for events
        if(!(*eq).empty())
        {
            Event toHandle = (*eq).top();
            // If the top event has the same time as the current time
            while(toHandle.getTime()==time&&!(*eq).empty())
            {
                // Popping the top event off
                (*eq).pop();
                // Getting the wire
                Wire* toUpdate = toHandle.getWire();
                // Setting its new value
                (*toUpdate).setValue(toHandle.getValue());
                // Updating its vector of gates
                priority = (*toUpdate).updateDriving(time, eq, priority);
                // Reloading the top event if the event queue is not empty
                if(!(*eq).empty())
                {
                    toHandle = (*eq).top();
                }
            }
        }

        // Poke each output
        for(int i=0; i<outputs.size(); i++)
        {
            // Poke function causes the output to print its current value to the output string
            (outputs[i]).poke();
        }
        // Runs three more cycles then quits if no more events to process
        if((*eq).empty())
        {
            for(int i=0; i<2; i++)
            {
                for(int j=0; j<outputs.size(); j++)
                {
                    (outputs[j]).poke();
                }
            }
            break;
        }

    }

    // Finding longest output for clean printing purposes
    int longest=0;
    for(int i=0; i<outputs.size(); i++)
    {
        int comp = (outputs[i].getTag()).size();
        if(comp>longest)
        {
            longest = comp;
        }
    }

    // Printing circuit name
    cout<<endl<<crctName<<endl;
```

```
463     // Printing outputs
464     for(int i=0; i<outputs.size(); i++)
465     {
466         string toPrint = outputs[i].getTag();
467         for(int i = toPrint.size();i<=longest;i++)
468         {
469             toPrint += " ";
470         }
471         cout<<toPrint;
472         outputs[i].print();
473     }
474     if(time!=numcycl)
475     {
476         cout<<endl<<endl<<"Note: Simulation stopped after "<<time<<" cycles due to no further activity.\n";
477     }
478     return 0;
479 }
```