```cpp
// See CXORGate.h
//
//--Includes-----------------------------------------------------------------
#include "CXORGate.h"

//---CXORGate Implementation--------------------------------------------------
CXORGate::CXORGate() : CLogic()
{
    mInputs = std::vector<eLogicLevel>(nInputs, LOGIC_UNDEFINED);
    mOutputs = std::vector<eLogicLevel>(nOutputs, LOGIC_UNDEFINED);
    mpOutputConnections = std::vector<CWire*>(nOutputs, NULL);
    ComputeOutput();
}

void CXORGate::ComputeOutput()
{
    // XOR logic
    if (mInputs[0] == LOGIC_UNDEFINED || mInputs[1] == LOGIC_UNDEFINED)
    {
        mOutputs[0] = LOGIC_UNDEFINED;
    }
    else if ((mInputs[0] == LOGIC_HIGH || mInputs[1] == LOGIC_HIGH) && mInputs[0] != mInputs[1])
    {
        mOutputs[0] = LOGIC_HIGH;
    }
    else
    {
        mOutputs[0] = LOGIC_LOW;
    }
    // Drive output
    if (mpOutputConnections[0] != NULL) mpOutputConnections[0]->DriveLevel(mOutputs[0]);
}
```

```
    1: #ifndef _CCIRCUIT_H
    2: #define _CCIRCUIT_H
    3:
    4: //--Includes-------------------------------------------------------------------
    5: #include "CLogic.h"
    6:
    7: #include <vector>
    8: #include <unordered_map>
    9: #include <tuple>
   10: #include <string>
   11:
   12: //---CCircuit Declaration--------------------------------------------------------
   13: // Subclass of CLogic, providing functionality to build combinatorial circuits using CCircuit or CGate obje
cts.
   14: class CCircuit: public CLogic
   15: {
   16: public:
   17:     /**
   18:      * Constructor
   19:      */
   20:     CCircuit();
   21:
   22:     /**
   23:      * Destuctor
   24:      */
   25:     ~CCircuit();
   26:
   27:     /**
   28:      * Connect wire to input of logic element
   29:      *
   30:      * @param wire wire to connect from
   31:      * @param logic gate to connect to
   32:      * @param input input of gate to connect to
   33:      */
   34:     void ConnectWireToLogic(std::string wire, std::string logic, int input);
   35:
   36:     /**
   37:      * Connect output of logic element to wire
   38:      *
   39:      * @param logic logic element to connect from
   40:      * @param output output of logic element to connect from
   41:      * @param wire wire to connect to
   42:      */
   43:     void ConnectLogicToWire(std::string logic, int output, std::string wire);
   44:
   45:     /**
   46:      * Add logic element to this CLogic instance
   47:      *
   48:      * @param logic name of logic element
   49:      * @param clogic pointer to logic element
   50:      */
   51:     void AddLogic(std::string logic, CLogic* clogic);
   52:
   53:     /**
   54:      * Add wire to this CLogic instance
   55:      *
   56:      * @param wire name of wire
   57:      */
   58:     void AddWire(std::string wire);
   59:
   60:     /**
   61:      * Connect the circuit input to wire
   62:      *
   63:      * @param wire name of wire
   64:      * @param circuitInput which input number to bind from
   65:      */
   66:     void MapInput(std::string wire, int circuitInput = -1);
   67:
   68:     /**
   69:      * Connect the output of logic element to circuit output
   70:      *
   71:      * @param logic name of logic element
   72:      * @param output output of logic element to connect from
   73:      * @param circuitOnput which output number to bind to
   74:      */
   75:     void MapOutput(std::string logic, int logicOutput, int circuitOutput = -1);
   76:
   77: private:
   78:     /**
   79:      * Compute the output levels of this Clogic object
   80:      */
   81:     void ComputeOutput();
   82:
```

```
83:     std::unordered_map<std::string, CLogic*> mLogics;               // gate pointers
84:     std::unordered_map<std::string, CWire*> mWires;                 // wire pointers
85:
86:     std::vector<std::tuple<int, std::string>> inputMap;             // input mapping
87:     std::vector<std::tuple<std::string, int, int>> outputMap;       // output mapping
88: };
89:
90: #endif
```

```cpp
// See TestDriver.h
//
//--Includes-----------------------------------------------------------------
#include "TestDriver.h"
#include "CANDGate.h"
#include "CORGate.h"
#include "CXORGate.h"
#include "CNOTGate.h"
#include "CCircuit.h"

#include <utility>
#include <vector>
#include <iostream>
#include <string>
#include <bitset>
#include <cmath>

//--TestDriver Implementation------------------------------------------------------------------
std::pair<std::string, CLogic*> TestDriver::NewCircuit () {

    std::string CircuitName = "Unknown circuit";
    CCircuit* Circuit = new CCircuit();

    while(true)
    {
        std::string Request;
        std::cin >> Request;  // get the next word from the input stream
        std::cout << "Processing input token: " << Request << std::endl;

        if( Request[0] == '#' )
        {
            // a comment line
            // get the rest of the line and ignore it
            std::string DummyVar;
            getline( std::cin, DummyVar );
        }
        else if( Request.compare( "component" ) == 0 )
        {
            std::string GateType;
            std::string GateName;
            std::cin >> GateType;
            std::cin >> GateName;
            std::cout << "Adding gate of type " << GateType << " named " << GateName << std::endl;
            // Allocate new gate of specified type
            CLogic* Gate;
            if (GateType.compare( "or" ) == 0)
            {
                Gate = new CORGate();
            }
            else if (GateType.compare( "and" ) == 0)
            {
                Gate = new CANDGate();
            }
            else if (GateType.compare( "xor" ) == 0)
            {
                Gate = new CXORGate();
            }
            else if (GateType.compare( "not" ) == 0)
            {
                Gate = new CNOTGate();
            }
            else
            {
                std::cout << "Unrecognised gate " << GateType << std::endl;
                std::cout << "Continuing to next line" << std::endl;
                // get the rest of the line and ignore it
                std::string DummyVar;
                getline( std::cin, DummyVar );
                continue;
            }
            // Add new gate to circuit
            Circuit->AddLogic(GateName, Gate);
        }

        else if( Request.compare( "wire" ) == 0 )
        {
            std::string GateName;
            std::string Input;
            std::string WireName;

            std::cin >> WireName;
            std::cin >> Input;
            std::cin >> GateName;
```

```
 84:
 85:             // Add new wire to circuit, and connect it
 86:             Circuit->AddWire(WireName);
 87:             Circuit->ConnectWireToLogic(WireName, GateName, stoi(Input));
 88:         }
 89:
 90:         else if( Request.compare( "connect" ) == 0 )
 91:         {
 92:             std::string GateName;
 93:             std::string Output;
 94:             std::string WireName;
 95:
 96:             std::cin >> GateName;
 97:             std::cin >> Output;
 98:             std::cin >> WireName;
 99:
100:             // Connect specified output of named gate to named wire
101:             Circuit->ConnectLogicToWire(GateName, stoi(Output), WireName);
102:         }
103:
104:         else if( Request.compare( "testerOutput" ) == 0 )
105:         {
106:             std::string GateName;
107:             std::string Output;
108:
109:             std::cin >> GateName;
110:             std::cin >> Output;
111:
112:             // Set specified output of named gate as output
113:             Circuit->MapOutput(GateName, stoi(Output));
114:         }
115:
116:         else if( Request.compare( "testerInput" ) == 0 )
117:         {
118:             std::string WireName;
119:             std::cin >> WireName;
120:
121:             // Set named wire as input
122:             Circuit->MapInput(WireName);
123:         }
124:
125:         else if( Request.compare( "end" ) == 0 )
126:         {
127:             // Save circuit name and stop reading file
128:             std::cin >> CircuitName;
129:             break;
130:         }
131:
132:         else
133:         {
134:             std::cout << "Unrecognised command " << Request << std::endl;
135:             std::cout << "Continuing to next line" << std::endl;
136:             // get the rest of the line and ignore it
137:             std::string DummyVar;
138:             getline( std::cin, DummyVar );
139:         }
140:     }
141:
142:     // Return circuit name and pointer
143:     return std::make_pair(CircuitName, Circuit);
144: }
145:
146: void TestDriver::TestCircuit (std::pair<std::string, CLogic*> &CircuitInfo, std::string &Input, int i)
147: {
148:     if (i >= CircuitInfo.second->InputSize())
149:     {
150:         std::string Name = CircuitInfo.first;
151:         auto Circuit = CircuitInfo.second;
152:         const std::size_t InputWidth = Circuit->InputSize();
153:         const std::size_t OutputWidth = Circuit->OutputSize();
154:
155:         // Drive each input with corressponding assignment
156:         for (int j = 0; j < int(InputWidth); j++){
157:             Circuit->DriveInput(j, (Input[j] == '1') ? LOGIC_HIGH : LOGIC_LOW);
158:         }
159:
160:         // Get all outputs and print
161:         std::string Output = "";
162:         for (int j = 0; j < int(OutputWidth); j++){
163:             if (Circuit->GetOutputState(j) == LOGIC_HIGH)
164:             {
165:                 Output.push_back('1');
166:             }
```

```
  167:             else if (Circuit->GetOutputState(j) == LOGIC_LOW)
  168:             {
  169:                 Output.push_back('0');
  170:             }
  171:             else {
  172:                 Output.push_back('Z');
  173:             }
  174:         }
  175:
  176:         std::cout << "[" << Name << "]"
  177:                     << " Input: " << Input
  178:                     << " >>> "
  179:                     << " Output: " << Output
  180:                     << std::endl;
  181:     }
  182:     else
  183:     {
  184:         // Recursion
  185:         Input.push_back('0');
  186:         TestCircuit(CircuitInfo, Input, i+1);
  187:         Input.pop_back();
  188:
  189:         Input.push_back('1');
  190:         TestCircuit(CircuitInfo, Input, i+1);
  191:         Input.pop_back();
  192:     }
  193:     return;
  194: }
```

```
 1: #ifndef _CXORGATE_H
 2: #define _CXORGATE_H
 3:
 4: //--Includes-----------------------------------------------------------------
 5: #include "CLogic.h"
 6:
 7: //---CXORGate Declaration-------------------------------------------------------
 8: // Subclass of CLogic that simulates a XOR gate
 9: class CXORGate: public CLogic
10: {
11: public:
12:     /**
13:      * Constructor
14:     */
15:     CXORGate();
16:
17: private:
18:     /**
19:      * Compute the output levels of this Clogic object
20:     */
21:     void ComputeOutput();
22:
23:     static const int nInputs = 2;       // number of inputs for a XOR gate
24:     static const int nOutputs = 1;      // number of outputs for a XOR gate
25: };
26:
27: #endif
```

```cpp
// See CLogic.h
//
//--Includes-----------------------------------------------------------------
#include "CCircuit.h"

//---CCircuit Implementation--------------------------------------------------

CCircuit::CCircuit():CLogic(){}

CCircuit::~CCircuit()
{
    // Delete all dynamically allocated wires and logic elements
    for (std::pair<std::string, CWire*> p : mWires) delete p.second;
    for (std::pair<std::string, CLogic*> p : mLogics) delete p.second;
}

void CCircuit::ConnectWireToLogic(std::string wire, std::string logic, int input)
{
    mWires[wire]->AddOutputConnection(mLogics[logic], input);
}

void CCircuit::ConnectLogicToWire(std::string logic, int output, std::string wire)
{
    mLogics[logic]->ConnectOutput(output, mWires[wire]);
}

void CCircuit::AddLogic(std::string logic, CLogic* clogic)
{
    // Add logic if its name is not already used
    if (mLogics.find(logic) == mLogics.end())
    {
        mLogics[logic] = clogic;
    }
}

void CCircuit::AddWire(std::string wire)
{
    // Add wire if its name is not already used
    if (mWires.find(wire) == mWires.end())
    {
        mWires[wire] = new CWire();
    }
}

void CCircuit::MapInput(std::string wire, int circuitInput)
{
    // If circuitInput is default value, set it as a new input
    if (circuitInput < 0){
        circuitInput = mInputs.size();
    }

    // Expand inputs
    while(int(mInputs.size()) < circuitInput + 1){
        mInputs.push_back(LOGIC_UNDEFINED);
    }

    // Add mapping
    inputMap.push_back(std::make_tuple(
        circuitInput,
        wire
        ));
}

void CCircuit::MapOutput(std::string logic, int logicOutput, int circuitOutput)
{
    // If circuitOutput is default value, set it as a new output
    if (circuitOutput < 0){
        circuitOutput = mOutputs.size();
    }

    // Expand outputs
    while(int(mOutputs.size()) < circuitOutput + 1){
        mOutputs.push_back(LOGIC_UNDEFINED);
        mpOutputConnections.push_back(NULL);
    }

    // Add mapping
    outputMap.push_back(std::make_tuple(
        logic,
        logicOutput,
        circuitOutput
        ));
}
```

```
 84:
 85: void CCircuit::ComputeOutput()
 86: {
 87:     // Look through input mapping and drive all inputs.
 88:     for (std::tuple<int, std::string> t : inputMap)
 89:     {
 90:         mWires[std::get<1>(t)]->DriveLevel(mInputs[std::get<0>(t)]);
 91:     }
 92:
 93:     // Look through output mapping and drive all outputs.
 94:     for (std::tuple<std::string, int, int> t : outputMap)
 95:     {
 96:         eLogicLevel output = mLogics[std::get<0>(t)]->GetOutputState(std::get<1>(t));
 97:         mOutputs[std::get<2>(t)] = output;
 98:         if (mpOutputConnections[std::get<2>(t)] != NULL)
 99:         {
100:             mpOutputConnections[std::get<2>(t)]->DriveLevel(output);
101:         }
102:     }
103: }
```

```cpp
// See CANDGate.h
//
//--Includes-----------------------------------------------------------------
#include "CANDGate.h"

//---CANDGate Implementation--------------------------------------------------
CANDGate::CANDGate() : CLogic()
{
    mInputs = std::vector<eLogicLevel>(nInputs, LOGIC_UNDEFINED);
    mOutputs = std::vector<eLogicLevel>(nOutputs, LOGIC_UNDEFINED);
    mpOutputConnections = std::vector<CWire*>(nOutputs, NULL);
    ComputeOutput();
}

void CANDGate::ComputeOutput()
{
    // AND logic
    if (mInputs[0] == LOGIC_UNDEFINED || mInputs[1] == LOGIC_UNDEFINED)
    {
        mOutputs[0] = LOGIC_UNDEFINED;
    }
    else if (mInputs[0] == LOGIC_HIGH && mInputs[1] == LOGIC_HIGH)
    {
        mOutputs[0] = LOGIC_HIGH;
    }
    else
    {
        mOutputs[0] = LOGIC_LOW;
    }
    // Drive output
    if (mpOutputConnections[0] != NULL) mpOutputConnections[0]->DriveLevel(mOutputs[0]);
}
```

```
 1: #ifndef _CNOTGATE_H
 2: #define _CNOTGATE_H
 3:
 4: //--Includes-----------------------------------------------------------------
 5: #include "CLogic.h"
 6:
 7: //---CNOTGate Declaration-------------------------------------------------------
 8: // Subclass of CLogic that simulates a NOT gate
 9: class CNOTGate: public CLogic
10: {
11: public:
12:     /**
13:      * Constructor
14:     */
15:     CNOTGate();
16:
17: private:
18:     /**
19:      * Compute the output levels of this Clogic object
20:     */
21:     void ComputeOutput();
22:
23:     static const int nInputs = 1;       // number of inputs for a NOT gate
24:     static const int nOutputs = 1;      // number of inputs for a NOT gate
25: };
26:
27: #endif
```

```cpp
// See CORGate.h
//
//--Includes-----------------------------------------------------------------
#include "CORGate.h"

//---CORGate Implementation---------------------------------------------------
CORGate::CORGate() : CLogic()
{
    mInputs = std::vector<eLogicLevel>(nInputs, LOGIC_UNDEFINED);
    mOutputs = std::vector<eLogicLevel>(nOutputs, LOGIC_UNDEFINED);
    mpOutputConnections = std::vector<CWire*>(nOutputs, NULL);
    ComputeOutput();
}

void CORGate::ComputeOutput()
{
    // XOR logic
    if (mInputs[0] == LOGIC_UNDEFINED || mInputs[1] == LOGIC_UNDEFINED)
    {
        mOutputs[0] = LOGIC_UNDEFINED;
    }
    else if (mInputs[0] == LOGIC_HIGH || mInputs[1] == LOGIC_HIGH)
    {
        mOutputs[0] = LOGIC_HIGH;
    }
    else
    {
        mOutputs[0] = LOGIC_LOW;
    }
    // Drive output
    if (mpOutputConnections[0] != NULL) mpOutputConnections[0]->DriveLevel(mOutputs[0]);
}
```

```
 1: #ifndef _CANDGATE_H
 2: #define _CANDGATE_H
 3:
 4: //--Includes-----------------------------------------------------------------
 5: #include "CLogic.h"
 6:
 7: //---CANDGate Declaration-------------------------------------------------------
 8: // Subclass of CLogic that simulates an AND gate
 9: class CANDGate: public CLogic
10: {
11: public:
12:     /**
13:      * Constructor
14:     */
15:     CANDGate();
16:
17: private:
18:     /**
19:      * Compute the output levels of this Clogic object
20:     */
21:     void ComputeOutput();
22:
23:     static const int nInputs = 2;       // number of inputs for an AND gate
24:     static const int nOutputs = 1;      // number of outputs for an AND gate
25: };
26:
27: #endif
```

```
    1: #ifndef _TESTDRIVER_H
    2: #define _TESTDRIVER_H
    3:
    4: //--Includes-------------------------------------------------------------------
    5: #include "CLogic.h"
    6:
    7: #include <string>
    8:
    9: //---TestDriver Declaration-----------------------------------------------------
   10: //
   11: // Testdriver for combinatorial logic circuit (CLC) simulator
   12: //
   13: // Provides classes and methods to create and test combinatorial circuits.
   14: //
   15: // CLogic is a template of a logic cell: any gate or circuit of gates.
   16: // CLogic units can be connected with CWires inside CCircuits and driven to get simulated outputs.
   17: //
   18: // CGates and CCircuit implement CLogic.
   19: // CGates simulate simple logic gates with hard coded behaviour.
   20: // CCircuits simulate complex combinatorial circuits, containing any number of CCircuits and CGates connect
ed together.
   21: //
   22: // Circuit accepts CLCs defined by .circuit files, piped directly into the executable.
   23: // Each .circuit file shall define only one CLC.
   24: // Syntax:
   25: //                  Command                                |               Definition
   26: //    ____________________________________________________|_____________________________________________________
_
   27: //
   28: //      component {gateType} {gateName}                 > "component" declares new component of type
   29: //                                                        {gateType}[and, or, xor, not] and name {gateName}
   30: //
   31: //      wire {wireName} {inputNo} {gateName}            > "wire" declares new wire {wireName} if it doesnt ex
ist
   32: //                                                        and connects it to input {inputNo} of gate {gateN
ame}
   33: //
   34: //      connect {gateName} {outputNo} {wireName}        > "connect" connects output {inputNo} of gate
   35: //                                                        {gateName} to wire {wireName}
   36: //
   37: //      testerOutput {gateName} {outputNo}              > "testerOutput" adds output {inputNo} of gate
   38: //                                                        {gateName} as an output of the whole circuit.
   39: //
   40: //      testerInput {wireName} {outputNo}               > "testerInput" adds wire {wireName} as an output
   41: //                                                        of the whole circuit.
   42: //
   43: //      end {circuitName}                               > "end" signifies the end of the declaration for
   44: //                                                        circuit with name {circuitName}
   45: //
   46: // Copyright (c) Daniel Shen 2023
   47:
   48: class TestDriver
   49: {
   50:   public:
   51:     /**
   52:      * Creates a new circuit from a .circuit file piped to cin.
   53:      *
   54:      * @return pair containing circuit name and circuit object pointer
   55:     */
   56:     std::pair<std::string, CLogic*>  NewCircuit ();
   57:
   58:     /**
   59:      * Prints truth table for a circuit. Recursively finds all possible assignments.
   60:      *
   61:      * @param CircuitInfo pair containing circuit name and circuit object pointer
   62:      * @param Input empty non-const string buffer used in recursion.
   63:      *              Initialize an empty string to a variable and pass as this argument when calling.
   64:      * @param i integer argument used in recursion. Don't provide this when calling.
   65:     */
   66:     void TestCircuit (std::pair<std::string, CLogic*> &CircuitInfo, std::string &Input, int i = 0);
   67:
   68:   private:
   69:     /**
   70:      * Private function for testing a particular assignment on a circuit.
   71:      *
   72:      * @param Name circuit name
   73:      * @param Circuit circuit pointer
   74:      * @param Input input assignment as a boolean number string
   75:     */
   76:     void TestInput (std::string Name, CLogic* Circuit, std::string Input);
   77:
   78: };
   79:
```

```
80: #endif
```

```
#ifndef _CWIRE_H
#define _CWIRE_H

//--Forward Declaration
class CLogic;

//--Consts and enums-----------------------------------------------------------
enum eLogicLevel // enum defining the possible states of a logic line
{
    LOGIC_UNDEFINED = -1,
    LOGIC_LOW = 0,
    LOGIC_HIGH = 1
};

//---CWire Declaration----------------------------------------------------------
// CWire is used to connect devices in this simulation
// A CWire has a single input, and may drive multiple outputs
// The global variable MaxFanout controls how many outputs each wire can have
// Each wire output drives a specific input of a specific gate
// The wire's input is controlled via the DriveLevel function
class CWire
{
  public:
    /**
     * Constructor
     */
    CWire();

    /**
     * Adds to the list of outputs that this wire drives
     *
     * @param apGateToDrive gate to drive
     * @param aGateInputToDrive input of gate to drive
     */
    void AddOutputConnection(CLogic *apGateToDrive, int aGateInputToDrive);

    /**
     * Drives the wire's value, so that each of its connected outputs
     *
     * @param aNewLevel levels to drive this wire with
     */
    void DriveLevel(eLogicLevel aNewLevel);

  private:
    static const int MaxFanout = 2;     // max gate inputs that one gate output can drive
    int mNumOutputConnections;            // how many outputs are connected
    CLogic *mpGatesToDrive[MaxFanout];    // list of connected gates
    int mGateInputIndices[MaxFanout];     // list of input to drive in each gate
};

#endif
```

```
#ifndef _CLOGIC_H
#define _CLOGIC_H

//--Includes-------------------------------------------------------------------
#include "CWire.h"

#include <vector>

//---Logic Declaration----------------------------------------------------------
// CLogic is a template class used to represent logic cells (i.e. gates or subcircuits)
class CLogic
{
  public:
    /**
     * Constructor
     */
    CLogic();

    /**
     * Destructor
     */
    virtual ~CLogic();

    /**
     * connect this logic element's output to a wire
     *
     * @param aOutputIndex output number
     * @param apOutputConnection wire to connect to
     */
    void ConnectOutput(int aOutputIndex, CWire *apOutputConnection);

    /**
     * drive the specified input of this logic element
     *
     * @param aInputIndex input number
     * @param aNewLevel level to drive input
     */
    void DriveInput(int aInputIndex, eLogicLevel aNewLevel);

    /**
     * return current output level of this logic element
     *
     * @param aOutputIndex output number
     */
    eLogicLevel GetOutputState(int aOutputIndex);

    /**
     * return number of inputs of this logic element
     *
     * @return number of inputs
     */
    int InputSize();

    /**
     * return number of outputs of this logic element
     *
     * @return number of outputs
     */
    int OutputSize();

  protected:

    /**
     * Pure virtual function
     *
     * For computing the output levels of Clogic objects
     */
    virtual void ComputeOutput() = 0;

    std::vector<eLogicLevel> mInputs;            // Input levels
    std::vector<eLogicLevel> mOutputs;           // Output levels
    std::vector<CWire*> mpOutputConnections;     // Output wires
};

#endif
```

```
#ifndef _CORGATE_H
#define _CORGATE_H

//--Includes-------------------------------------------------------------------
#include "CLogic.h"

//---CORGate Declaration--------------------------------------------------------
// Subclass of CLogic that simulates an OR gate
class CORGate: public CLogic
{
public:
    /**
     * Constructor
    */
    CORGate();

private:
    /**
     * Compute the output levels of this Clogic object
    */
    void ComputeOutput();

    static const int nInputs = 2;       // number of inputs for a OR gate
    static const int nOutputs = 1;      // number of outputs for a OR gate
};


#endif
```

```cpp
// Main
//
// Calls TestDriver class to test combinatorial logic circuits.
//
// Copyright (c) Daniel Shen 2023

//--Includes-----------------------------------------------------------------
#include "TestDriver.h"

#include <string>
#include <iostream>

//---Main---------------------------------------------------------------------
int main()
{
    // Create new testdriver
    TestDriver T = TestDriver();

    // Create new circuit
    auto CircuitInfo = T.NewCircuit();

    // Test circuit with all assignments
    std::string Assignment = "";
    T.TestCircuit(CircuitInfo, Assignment);

    // Delete circuit
    delete(CircuitInfo.second);

    return 0;
}
```

```cpp
// See CLogic.h
//
//--Includes-----------------------------------------------------------------
#include "CLogic.h"

//---CLogic Implementation----------------------------------------------------
CLogic::CLogic(){}

CLogic::~CLogic(){}

void CLogic::ConnectOutput(int aOutputIndex, CWire *apOutputConnection)
{
    // Connect new output and recompute outputs
    mpOutputConnections[aOutputIndex] = apOutputConnection;
    ComputeOutput();
}

void CLogic::DriveInput(int aInputIndex, eLogicLevel aNewLevel)
{
    // Connect new input and recompute outputs
    mInputs[aInputIndex] = aNewLevel;
    ComputeOutput();
}

eLogicLevel CLogic::GetOutputState(int aOutputIndex)
{
    // Resize outputs to required index
    while(int(mOutputs.size()) < aOutputIndex + 1)
    {
        mOutputs.push_back(LOGIC_UNDEFINED);
        mpOutputConnections.push_back(NULL);
    }
    return mOutputs[aOutputIndex];
}

int CLogic::InputSize()
{
    return mInputs.size();
}

int CLogic::OutputSize()
{
    return mOutputs.size();
}
```

```cpp
// See CWire.h
//
//--Includes-----------------------------------------------------------------
#include "CLogic.h"
#include "CWire.h"

//---CWire Implementation-----------------------------------------------------
CWire::CWire()
{
    mNumOutputConnections = 0;
}

void CWire::AddOutputConnection(CLogic *apGateToDrive, int aGateInputToDrive)
{
    mpGatesToDrive[mNumOutputConnections] = apGateToDrive;
    mGateInputIndices[mNumOutputConnections] = aGateInputToDrive;
    ++mNumOutputConnections;
}

void CWire::DriveLevel(eLogicLevel aNewLevel)
{
    // Drive each connected output
    for (int i = 0; i < mNumOutputConnections; ++i)
        mpGatesToDrive[i]->DriveInput(mGateInputIndices[i], aNewLevel);
}
```

```cpp
// See CNOTGate.h
//
//--Includes-----------------------------------------------------------------
#include "CNOTGate.h"

//---CNOTGate Implementation--------------------------------------------------
CNOTGate::CNOTGate() : CLogic()
{
    mInputs = std::vector<eLogicLevel>(nInputs, LOGIC_UNDEFINED);
    mOutputs = std::vector<eLogicLevel>(nOutputs, LOGIC_UNDEFINED);
    mpOutputConnections = std::vector<CWire*>(nOutputs, NULL);
    ComputeOutput();
}

void CNOTGate::ComputeOutput()
{
    // NOT logic
    if (mInputs[0] == LOGIC_HIGH)
    {
        mOutputs[0] = LOGIC_LOW;
    }
    else if (mInputs[0] == LOGIC_LOW)
    {
        mOutputs[0] = LOGIC_HIGH;
    }
    else
    {
        mOutputs[0] = LOGIC_UNDEFINED;
    }
    // Drive output
    if (mpOutputConnections[0] != NULL) mpOutputConnections[0]->DriveLevel(mOutputs[0]);
}
```