```
 1: #ifndef _COMPUTER_H
 2: #define _COMPUTER_H
 3:
 4:
 5: #include <string>
 6:
 7: class Person; // forward declaration
 8:
 9: //---Class definition for Computer---
10: class Computer
11: {
12:     public:
13:         void Init( std::string Name );
14:         void SendHello( Person* pDest );
15:         void Report();
16:
17:     private:
18:         std::string _name;
19: };
20:
21:
22: #endif
```

```cpp
#include <iostream>
#include <string>

#include "Person.h"
#include "Computer.h"

//---
int main()
{
    Person Alice; // Alice is an instance of the class Person
    Person Bob;   // Bob is a second instance
    Person Carol;

    Alice.Init( "Alice" );
    Bob.Init( "Bob" );
    Carol.Init( "Carol" );

    Alice.SendHello( &Bob );
    Alice.SendHello( &Carol );
    Bob.SendHello( &Carol );

    Bob.SendHello( &Bob );
    Alice.SendHello( &Alice );

    Computer Hal;
    Hal.Init( "Hal" );
    Hal.SendHello( &Carol );
}


```

```
 1: #ifndef _PERSON_H
 2: #define _PERSON_H
 3:
 4:
 5: #include <string>
 6:
 7: class Person
 8: {
 9:     public:
10:         void Init( std::string Name );
11:         void SendHello( Person* pDest );
12:         void ReceiveHello();
13:         void ReceiveHello( Person* pSource );
14:         void Report();
15:         std::string GetName() { return _name;}
16:
17:     private:
18:         int _helloCount;
19:         std::string _name;
20: };
21:
22:
23:
24:
25: #endif
26:
```

```cpp
#include <iostream>
#include <string>

#include "Person.h"

//---Class implementation for Person---
void Person::Init( std::string Name )
{
    _helloCount = 0;
    _name = Name;
    std::cout << "[Init]: " << _name << " is initialised with " << _helloCount << " hellos" << std::endl;
}

//---
void Person::SendHello( Person* pDest )
{
    std::cout << "[SendHello]: " << _name << " is saying hello to " << pDest->_name << std::endl;
    if( pDest == this )
        std::cout << "... that's a bit weird, not incrementing" << std::endl;
    else
        pDest->ReceiveHello( this );
    pDest->Report();
}

//---
void Person::Report()
{
    std::cout << "[Report]: " << _name << "'s _helloCount is now " << _helloCount << std::endl;
}

//---
void Person::ReceiveHello()
{
    std::cout << "[ReceiveHello]: " << _name << " is receiving hello"  << std::endl;
    _helloCount++;
    Report();
}

//---
void Person::ReceiveHello( Person* pSource )
{
    std::cout << "[ReceiveHello(p)]: " << _name << " is receiving hello from " << pSource->_name << std::en
dl;
    _helloCount++;
    std::cout << "[ReceiveHello(p)]: " << _name << " is saying hello back " << std::endl;
    pSource->ReceiveHello();
}


```

```cpp
#include <iostream>
#include <string>

#include "Computer.h"
#include "Person.h"


//---Class implementation for Person---
void Computer::Init( std::string Name )
{
    _name = Name;
    std::cout << "[Init]: " << _name << " is initialised" << std::endl;
}

//---
void Computer::SendHello( Person* pDest )
{
    std::cout << "[SendHello]: " << _name << " is saying hello to " << pDest->GetName() << std::endl;
    pDest->ReceiveHello();
    pDest->Report();
}


```

```cpp
// HelloClasses.cpp
//
// Example class with private members and public methods
// Each person keeps track of how many times they're said hello to
// A person can say hello to another person
//
// Initial revision: Donald G Dansereau, 2019
// Completed by:

#include <iostream>
#include <string>

//---Class definition for Person-------------------------------
// This defines the interface for this class
class Person
{
    public:
        // initialise a person with a given name
        void Init( std::string Name );

        // say hello to another person;
        // the second person is specified using a pointer
        void SendHello( Person* pDest );

    private:
        // The person's name
        std::string _name;

        // How many times someone's said hello to this person
        int _helloCount;
};

//-------------------------------------------------------------
int main()
{
    Person Alice; // Alice is an instance of the class Person
    Person Bob;   // Bob is another instance of Person

    Alice.Init( "Alice" );
    Bob.Init( "Bob" );

    // Let's introduce Alice and Bob
    Alice.SendHello( &Bob );
    Bob.SendHello( &Alice );
}


//---Class implementation for Person---------------------------
// Init zeroes the hello count, and assigns a name
// it also outputs to the screen so we can see what's going on
void Person::Init( std::string Name )
{
    _helloCount = 0;
    _name = Name;
    std::cout << "[Init]: " << _name << " is initialised with "
        << _helloCount << " hellos" << std::endl;
}

//-------------------------------------------------------------
// SendHello says hello to another person by incrementing their
// hello count.  It also outputs information to the screen so
// we can see what's going on.
void Person::SendHello( Person* pDest )
{
    std::cout << "[SendHello]: " << _name
        << " is saying hello to " << pDest->_name << std::endl;

    pDest->_helloCount++;

    std::cout << "[SendHello]: " << pDest->_name
        << "'s _helloCount is now " << pDest->_helloCount
        << std::endl;
}

```

```
    1: // A combinatorial logic circuit simulator
    2: //
    3: // The simulated circuit has logic gates and wires to connect them. Test
    4: // signals are generated to test the circuit's output, which is printed
    5: // to screen.
    6: //
    7: // Only NAND gates are supported, and the circuit topology is hard-coded.
    8: //
    9: // This code is functionally complete: it is capable of simulating any non-
   10: // recurrent combinatorial logic function. The provided example demonstrates
   11: // a 2-input, 2-gate, 1-output logic function.
   12: //
   13: // Parts of the code are well-designed and well-written, but some rules of
   14: // object-oriented design are broken, in particular the contents of main are
   15: // not well encapsulated in objects. There are also examples of poor coding
   16: // style including a lack of comments throughout.
   17: //
   18: // Copyright (c) Donald Dansereau, 2023
   19:
   20:
   21: //--Includes-----------------------------------------------------------------
   22: #include <iostream>
   23:
   24: //--Consts and enums---------------------------------------------------------
   25: const int InputsPerGate = 2;                  // number of inputs per nand gate
   26: const int MaxFanout = 2;                      // maximum fanout: max gate inputs that one gate output can dr
ive
   27:
   28: enum eLogicLevel                              // enum defining the possible states of a logic line
   29: {
   30:   LOGIC_UNDEFINED = -1,
   31:   LOGIC_LOW,
   32:   LOGIC_HIGH
   33: };
   34:
   35: //---Forward Declarations-----------------------------------------------------
   36: class CNandGate;  // forward declaration
   37:
   38: //---CWire Interface----------------------------------------------------------
   39: // CWire is used to connect devices in this simulation
   40: // A CWire has a single input, and may drive multiple outputs
   41: // The global variable MaxFanout controls how many outputs each wire can have
   42: // Each wire output drives a specific input of a specific gate
   43: // The wire's input is controlled via the DriveLevel function
   44: class CWire
   45: {
   46:   public:
   47:     void Init();
   48:
   49:     // AddOutputConnection adds to the list of outputs that this wire drives
   50:     // It accepts as parameters the nand gate whose input should be driven
   51:     // and the index specifying which of that gate's inputs should be driven.
   52:     void AddOutputConnection( CNandGate* apGateToDrive, int aGateInputToDrive );
   53:
   54:     // DriveLevel drives the wire's value, so that each of its connected outputs
   55:     // get set to the corresponding level
   56:     void DriveLevel( eLogicLevel aNewLevel );
   57:
   58:   private:
   59:     int mNumOutputConnections;            // how many outputs are connected
   60:     CNandGate* mpGatesToDrive[MaxFanout]; // list of connected gates
   61:     int mGateInputIndices[MaxFanout];     // list of input to drive in each gate
   62: };
   63:
   64: //---CNandGate Interface------------------------------------------------------
   65: class CNandGate
   66: {
   67:   public:
   68:     void Init();
   69:     void ConnectOutput( CWire* apOutputConnection );
   70:     void DriveInput( int aInputIndex, eLogicLevel aNewLevel );
   71:     eLogicLevel GetOutputState();
   72:
   73:   private:
   74:     void ComputeOutput();
   75:
   76:     eLogicLevel mInputs[ InputsPerGate ];
   77:     eLogicLevel mOutputValue;
   78:     CWire* mpOutputConnection;
   79: };
   80:
   81:
   82: //---main---------------------------------------------------------------------
```

```
 83: int main()
 84: {
 85:   const int NumNandGates = 2;                  // number of nand gates
 86:   const int NumWires = 3;                      // number of wires
 87:
 88:   CNandGate MyGates[NumNandGates];
 89:   CWire MyWires[NumWires];
 90:
 91:   for( int i=0; i<NumNandGates; ++i )
 92:     MyGates[i].Init();
 93:
 94:   for( int i=0; i<NumWires; ++i )
 95:     MyWires[i].Init();
 96:
 97:   // MyWires[0] and [1] are input wires for the circuit
 98:   MyWires[0].AddOutputConnection( &MyGates[0], 0 );
 99:   MyWires[0].AddOutputConnection( &MyGates[1], 0 );
100:   MyWires[1].AddOutputConnection( &MyGates[0], 1 );
101:
102:   // MyWires[2] is a connection between the output of MyGates[0] and MyGates[1] input 1
103:   MyWires[2].AddOutputConnection( &MyGates[1], 1 );
104:   MyGates[0].ConnectOutput( &MyWires[2] );
105:
106:   // Test each of the possible input states
107:   MyWires[0].DriveLevel( LOGIC_LOW );
108:   MyWires[1].DriveLevel( LOGIC_LOW );
109:   std::cout << "Testing input 0 0 result: " << MyGates[1].GetOutputState() << std::endl;
110:
111:   MyWires[0].DriveLevel( LOGIC_LOW );
112:   MyWires[1].DriveLevel( LOGIC_HIGH );
113:   std::cout << "Testing input 0 1 result: " << MyGates[1].GetOutputState() << std::endl;
114:
115:   MyWires[0].DriveLevel( LOGIC_HIGH );
116:   MyWires[1].DriveLevel( LOGIC_LOW );
117:   std::cout << "Testing input 1 0 result: " << MyGates[1].GetOutputState() << std::endl;
118:
119:   MyWires[0].DriveLevel( LOGIC_HIGH );
120:   MyWires[1].DriveLevel( LOGIC_HIGH );
121:   std::cout << "Testing input 1 1 result: " << MyGates[1].GetOutputState() << std::endl;
122:
123:   return 0;
124: }
125:
126: //---CWire Implementation-------------------------------------------------------
127: void CWire::Init()
128: {
129:   mNumOutputConnections = 0;
130: }
131: //---
132: void CWire::AddOutputConnection( CNandGate* apGateToDrive, int aGateInputToDrive )
133: {
134:   mpGatesToDrive[mNumOutputConnections] = apGateToDrive;
135:   mGateInputIndices[mNumOutputConnections] = aGateInputToDrive;
136:   ++mNumOutputConnections;
137: }
138: //---
139: void CWire::DriveLevel( eLogicLevel aNewLevel )
140: {
141:   for( int i=0; i<mNumOutputConnections; ++i )
142:     mpGatesToDrive[i]->DriveInput( mGateInputIndices[i], aNewLevel );
143: }
144:
145: //---CNandGate Implementation---------------------------------------------------
146: void CNandGate::Init()
147: {
148:   mInputs[0] = mInputs[1] = LOGIC_UNDEFINED;
149:   mpOutputConnection = NULL;
150:   ComputeOutput();
151: }
152: //---
153: void CNandGate::ConnectOutput( CWire* apOutputConnection )
154: {
155:   mpOutputConnection = apOutputConnection;
156: }
157: //---
158: void CNandGate::DriveInput( int aInputIndex, eLogicLevel aNewLevel )
159: {
160:   mInputs[aInputIndex] = aNewLevel;
161:   ComputeOutput();
162: }
163: //---
164: eLogicLevel CNandGate::GetOutputState()
165: {
```

```
166:   return mOutputValue;
167: }
168: //---
169: void CNandGate::ComputeOutput()
170: {
171:   eLogicLevel NewVal = LOGIC_HIGH;
172:   if( mInputs[0] == LOGIC_UNDEFINED || mInputs[1] == LOGIC_UNDEFINED )
173:     NewVal = LOGIC_UNDEFINED;
174:   else if( mInputs[0] == LOGIC_HIGH && mInputs[1] == LOGIC_HIGH )
175:     NewVal = LOGIC_LOW;
176:   mOutputValue = NewVal;
177:
178:   if( mpOutputConnection != NULL )
179:     mpOutputConnection->DriveLevel( mOutputValue );
180: }
181:
182:
```

```cpp
// HelloClasses.cpp
//
// Example class with private members and public methods
// Each person keeps track of how many times they're said hello to
// A person can say hello to another person
// Computers say hello to people, but not vice versa
//
// Initial revision: Donald G Dansereau, 2019
// Completed by:

#include <iostream>
#include <string>

//---Forward declarations-------------------------------------
class Person;

//---Class definition for Computer---------------------------
class Computer
{
    public:
        // initialise a computer with a given name
        void Init( std::string Name );

        // say hello to a person;
        // the second person is specified using a pointer
        void SendHello( Person* pDest );

        // report status
        void Report();

    private:
        // The computer's name
        std::string _name;
};


//---Class definition for Person-----------------------------
// This defines the interface for this class
class Person
{
    public:
        // initialise a person with a given name
        void Init( std::string Name );

        // say hello to another person;
        // the second person is specified using a pointer
        void SendHello( Person* pDest );

        // receive a hello
        void ReceiveHello();
        void ReceiveHello( Person* pSource );

        // report status
        void Report();

    private:
        // The person's name
        std::string _name;

        // How many times someone's said hello to this person
        int _helloCount;
};




//-------------------------------------------------------------
int main()
{
    Person Alice;
    Person Bob;
    Person Carol;

    Computer Hal;

    Alice.Init( "Alice" );
    Bob.Init( "Bob" );
    Carol.Init( "Carol" );
    Hal.Init( "Hal" );

    // Let's introduce Alice and Bob and Carol
    Alice.SendHello( &Bob );
    Bob.SendHello( &Carol );
```

```
   84:     Carol.SendHello( &Alice );
   85:
   86:     // Alice and bob greet themselves
   87:     Alice.SendHello( &Alice );
   88:     Bob.SendHello( &Bob );
   89:
   90:     // Hal says hi
   91:     Hal.SendHello( &Carol );
   92: }
   93:
   94:
   95: //---Class implementation for Person---------------------------
   96: // Init zeroes the hello count, and assigns a name
   97: // it also outputs to the screen so we can see what's going on
   98: void Person::Init( std::string Name )
   99: {
  100:     _helloCount = 0;
  101:     _name = Name;
  102:     std::cout << "[Init]: ";
  103:     Report();
  104: }
  105:
  106: //-------------------------------------------------------------
  107: // SendHello says hello to another person by calling their ReceiveHello
  108: // It also outputs information to the screen so we can see what's going on.
  109: void Person::SendHello( Person* pDest )
  110: {
  111:     std::cout << "[SendHello]: " << _name
  112:         << " is saying hello to " << pDest->_name << std::endl;
  113:
  114:     if( pDest == this )
  115:     {
  116:         std::cout << "... trying to say hello to themselves, disallowing" << std::endl;
  117:     }
  118:     else
  119:     {
  120:         pDest->ReceiveHello( this );
  121:     }
  122:
  123:     std::cout << "[SendHello]: ";
  124:     Report();
  125: }
  126:
  127:
  128: //-------------------------------------------------------------
  129: // Report: report name and hello count to std::cout
  130: void Person::Report()
  131: {
  132:     std::cout << _name << " has "
  133:         << _helloCount << " hellos" << std::endl;
  134: }
  135:
  136: //-------------------------------------------------------------
  137: // ReceiveHello: increment _helloCount
  138: void Person::ReceiveHello()
  139: {
  140:     ++_helloCount;
  141:     std::cout << "[ReceiveHello]: ";
  142:     Report();
  143: }
  144:
  145: //-------------------------------------------------------------
  146: // ReceiveHello: increment _helloCount and reciprocate
  147: void Person::ReceiveHello( Person* pSource )
  148: {
  149:     std::cout << "[ReceiveHello]: " << _name
  150:         << " is receiving hello from " << pSource->_name << std::endl;
  151:
  152:     ++_helloCount;
  153:     std::cout << "[ReceiveHello]: ";
  154:     Report();
  155:
  156:     std::cout << "... saying hello back: " << std::endl;
  157:     pSource->ReceiveHello();
  158: }
  159:
  160: //---Computer-----------------------------------------------------------------
  161: // Init assigns a name, and outputs to the screen so we can see what's going on
  162: void Computer::Init( std::string Name )
  163: {
  164:     _name = Name;
  165:     std::cout << "[Init]: ";
  166:     Report();
```

```
167: }
168:
169: //-------------------------------------------------------------
170: // SendHello says hello to a person by calling their ReceiveHello
171: // It also outputs information to the screen so we can see what's
172: // going on.
173: void Computer::SendHello( Person* pDest )
174: {
175:     std::cout << "[SendHello]: " << _name
176:         << " is saying hello to a person" << std::endl;
177:
178:     pDest->ReceiveHello();
179:
180:     std::cout << "[SendHello]: ";
181:     Report();
182: }
183:
184:
185: //-------------------------------------------------------------
186: // Report: report name to std::cout
187: void Computer::Report()
188: {
189:     std::cout << _name << " (computer) reporting" << std::endl;
190: }
```

```cpp
// HelloClasses.cpp
//
// Example class with private members and public methods
// Each person keeps track of how many times they're said hello to
// A person can say hello to another person
//
// Initial revision: Donald G Dansereau, 2019
// Completed by:

#include <iostream>
#include <string>

//---Class definition for Person-------------------------------
// This defines the interface for this class
class Person
{
    public:
        // initialise a person with a given name
        void Init( std::string Name );

        // say hello to another person;
        // the second person is specified using a pointer
        void SendHello( Person* pDest );

        // receive a hello
        void ReceiveHello();
        void ReceiveHello( Person* pSource );

        // report status
        void Report();

    private:
        // The person's name
        std::string _name;

        // How many times someone's said hello to this person
        int _helloCount;
};

//-------------------------------------------------------------
int main()
{
    Person Alice; // Alice is an instance of the class Person
    Person Bob;   // Bob is another instance of Person
    Person Carol; // Carol is another instance of Person

    Alice.Init( "Alice" );
    Bob.Init( "Bob" );
    Carol.Init( "Carol" );

    // Let's introduce Alice and Bob and Carol
    Alice.SendHello( &Bob );
    Bob.SendHello( &Carol );
    Carol.SendHello( &Alice );

    // Alice and bob greet themselves
    Alice.SendHello( &Alice );
    Bob.SendHello( &Bob );

}


//---Class implementation for Person---------------------------
// Init zeroes the hello count, and assigns a name
// it also outputs to the screen so we can see what's going on
void Person::Init( std::string Name )
{
    _helloCount = 0;
    _name = Name;
    std::cout << "[Init]: ";
    Report();
}

//-------------------------------------------------------------
// SendHello says hello to another person by incrementing their
// hello count.  It also outputs information to the screen so
// we can see what's going on.
void Person::SendHello( Person* pDest )
{
    std::cout << "[SendHello]: " << _name
        << " is saying hello to " << pDest->_name << std::endl;

    if( pDest == this )
```

```
84:     {
85:         std::cout << "... trying to say hello to themselves, disallowing" << std::endl;
86:     }
87:     else
88:     {
89:         pDest->ReceiveHello( this );
90:     }
91:
92:     std::cout << "[SendHello]: ";
93:     Report();
94: }
95:
96:
97: //-----------------------------------------------------------
98: // Report: report name and hello count to std::cout
99: void Person::Report()
100: {
101:     std::cout << _name << " has "
102:         << _helloCount << " hellos" << std::endl;
103: }
104:
105: //-----------------------------------------------------------
106: // ReceiveHello: increment _helloCount
107: void Person::ReceiveHello()
108: {
109:     ++_helloCount;
110:     std::cout << "[ReceiveHello]: ";
111:     Report();
112: }
113:
114: //-----------------------------------------------------------
115: // ReceiveHello: increment _helloCount
116: void Person::ReceiveHello( Person* pSource )
117: {
118:     std::cout << "[ReceiveHello]: " << _name
119:         << " is receiving hello from " << pSource->_name << std::endl;
120:
121:     ++_helloCount;
122:     std::cout << "[ReceiveHello]: ";
123:     Report();
124:
125:     std::cout << "... saying hello back: " << std::endl;
126:     pSource->ReceiveHello();
127: }
```

```cpp
// HelloClasses.cpp
//
// Example class with private members and public methods
// Each person keeps track of how many times they're said hello to
// A person can say hello to another person
//
// Initial revision: Donald G Dansereau, 2019
// Completed by:

#include <iostream>
#include <string>

//---Class definition for Person------------------------------
// This defines the interface for this class
class Person
{
    public:
        // initialise a person with a given name
        void Init( std::string Name );

        // say hello to another person;
        // the second person is specified using a pointer
        void SendHello( Person* pDest );

        // receive a hello
        void ReceiveHello();

        // report status
        void Report();

    private:
        // The person's name
        std::string _name;

        // How many times someone's said hello to this person
        int _helloCount;
};

//------------------------------------------------------------
int main()
{
    Person Alice; // Alice is an instance of the class Person
    Person Bob;   // Bob is another instance of Person
    Person Carol; // Carol is another instance of Person

    Alice.Init( "Alice" );
    Bob.Init( "Bob" );
    Carol.Init( "Carol" );

    // Let's introduce Alice and Bob and Carol
    Alice.SendHello( &Bob );
    Alice.SendHello( &Carol );
    Bob.SendHello( &Alice );
    Bob.SendHello( &Carol );
    Carol.SendHello( &Alice );
    Carol.SendHello( &Bob );

    // Alice and bob greet themselves
    Alice.SendHello( &Alice );
    Bob.SendHello( &Bob );
}


//---Class implementation for Person--------------------------
// Init zeroes the hello count, and assigns a name
// it also outputs to the screen so we can see what's going on
void Person::Init( std::string Name )
{
    _helloCount = 0;
    _name = Name;
    std::cout << "[Init]: ";
    Report();
}

//------------------------------------------------------------
// SendHello says hello to another person by incrementing their
// hello count.  It also outputs information to the screen so
// we can see what's going on.
void Person::SendHello( Person* pDest )
{
    std::cout << "[SendHello]: " << _name
        << " is saying hello to " << pDest->_name << std::endl;

```

```
 84:     pDest->ReceiveHello();
 85:
 86:     std::cout << "[SendHello]: ";
 87:     Report();
 88: }
 89:
 90:
 91: //-------------------------------------------------------------
 92: // Report: report name and hello count to std::cout
 93: void Person::Report()
 94: {
 95:     std::cout << _name << " has "
 96:         << _helloCount << " hellos" << std::endl;
 97: }
 98:
 99: //-------------------------------------------------------------
100: // ReceiveHello: increment _helloCount
101: void Person::ReceiveHello()
102: {
103:     ++_helloCount;
104:     std::cout << "[ReceiveHello]: ";
105:     Report();
106: }
107:
```

```
1: // ProcessingRobot.h
2: //
3: // Header file for a robot that processes items off a conveyor belt
4: // Initial revision: Donald G Dansereau, 2019
5: // Completed by:
6:
7: #ifndef _PROCESSINGROBOT_H
8: #define _PROCESSINGROBOT_H
9:
10: #include "Conveyor.h"
11:
12:
13: //-----------------------------------------------------------------
14: // Simulate a processing robot that removes items from a conveyor belt.
15: // Note that Init sets a pointer to the conveyor the robot will use.
16: class ProcessingRobot
17: {
18:     public:
19:         void Init( Conveyor* WhichConveyor, int CapacityItemsPerCycle );
20:         void ProcessItems();
21:         void Report();
22:
23:     private:
24:         Conveyor* _Conveyor;
25:         int _CapacityItemsPerCycle;
26:         int _TotNumProcessed;
27:         int _TotCapacity;
28:
29: };
30:
31: #endif
```

```cpp
// LoadingRobot.cpp
//
// Implementation file for a robot that can load items onto a conveyor belt
// Initial revision: Donald G Dansereau, 2019
// Completed by:


#include <cstdlib>      // rand

#include "LoadingRobot.h"
#include "Conveyor.h"


//-----------------------------------------------------------------
void LoadingRobot::Init( Conveyor* WhichConveyor )
{
    _Conveyor = WhichConveyor;
}

//-----------------------------------------------------------------
void LoadingRobot::AddItems()
{
    _Conveyor->AddItems( rand() % 10 );
}

```

```cpp
// Conveyor.cpp
//
// Implementation file for a simulated conveyor belt
// Initial revision: Donald G Dansereau, 2019
// Completed by:

#include <iostream>     // std::cout
#include <algorithm>    // std::max

#include "Conveyor.h"

//-----------------------------------------------------------------
void Conveyor::Init()
{
    _NumItemsOnConveyor = 0;
}

//-----------------------------------------------------------------
void Conveyor::AddItems( int n )
{
    _NumItemsOnConveyor += n;
}

//-----------------------------------------------------------------
int Conveyor::RemoveItems( int n )
{
    // Note that we cannot have a negative number of items on belt
    int NumRemoved = std::min( _NumItemsOnConveyor, n );
    _NumItemsOnConveyor -= NumRemoved;
    return NumRemoved;
}

//-----------------------------------------------------------------
void Conveyor::Report()
{
    std::cout << "Items on conveyor: " << _NumItemsOnConveyor << std::endl;
}

```

```cpp
// ProcessingRobot.cpp
//
// Implementation file for a robot that processes items off a conveyor belt
// Initial revision: Donald G Dansereau
// Completed by:

#include <iostream>
#include <cstdlib>      // rand
#include <algorithm>    // std::max

#include "ProcessingRobot.h"

//-------------------------------------------------------------
void ProcessingRobot::Init( Conveyor* WhichConveyor, int CapacityItemsPerCycle )
{
    _Conveyor = WhichConveyor;
    _CapacityItemsPerCycle = CapacityItemsPerCycle;
    _TotNumProcessed = 0;
    _TotCapacity = 0;
}

//-------------------------------------------------------------
void ProcessingRobot::ProcessItems()
{
    int NumProcessed = _Conveyor->RemoveItems( _CapacityItemsPerCycle );
    _TotNumProcessed += NumProcessed;
    _TotCapacity += _CapacityItemsPerCycle;

    std::cout << "Processed " << NumProcessed << " items" << std::endl;
}

//-------------------------------------------------------------
void ProcessingRobot::Report()
{
    double PercentUtilisation = double(_TotNumProcessed) * 100.0 / double(_TotCapacity);
    std::cout << "Total processed: " << _TotNumProcessed << ", utilisation: " << PercentUtilisation << "%"
<< std::endl;
}
```

```
 1: // Conveyor.h
 2: //
 3: // Header file for a simulated conveyor belt
 4: // Initial revision: Donald G Dansereau, 2019
 5: // Completed by:
 6:
 7: #ifndef _CONVEYOR_H
 8: #define _CONVEYOR_H
 9:
10: //-------------------------------------------------------------
11: // Simulate a conveyor belt. This version just counts how many
12: // objects are on the belt, and accepts requests to add and remove
13: // objects.
14: class Conveyor
15: {
16:     public:
17:         void Init();
18:         void AddItems( int n );
19:         int RemoveItems( int n ); // returns number actually removed
20:         void Report();
21:
22:     private:
23:         int _NumItemsOnConveyor;
24: };
25:
26:
27:
28: #endif
```

```
 1: // main.cpp
 2: //
 3: // Main file for simulated conveyor belt
 4: // Initial revision: Donald G Dansereau, 2019
 5: // Completed by:
 6:
 7: #include <iostream>
 8: #include <cstdlib>      // rand
 9: #include <algorithm>    // std::max
10:
11: #include "Conveyor.h"
12: #include "LoadingRobot.h"
13: #include "ProcessingRobot.h"
14:
15:
16: int main()
17: {
18:     Conveyor myConveyor;
19:     LoadingRobot myLoader;
20:     ProcessingRobot myProcessor;
21:
22:     myConveyor.Init();
23:     myLoader.Init( &myConveyor );
24:
25:     const int ProcessorCapacity_ItemsPerCycle = 5;
26:     myProcessor.Init( &myConveyor, ProcessorCapacity_ItemsPerCycle );
27:
28:     while( 1 )
29:     {
30:         myLoader.AddItems();
31:         myConveyor.Report();
32:         myProcessor.ProcessItems();
33:         myProcessor.Report();
34:         myConveyor.Report();
35:     }
36: }
37:
38:
```

```
 1: // LoadingRobot.h
 2: //
 3: // Header file for a robot that can load items onto a conveyor belt
 4: // Initial revision: Donald G Dansereau, 2019
 5: // Completed by:
 6:
 7: #ifndef _LOADINGROBOT_H
 8: #define _LOADINGROBOT_H
 9:
10: #include "Conveyor.h"
11:
12:
13: //-----------------------------------------------------------------
14: // Simulate a loading robot that places items on a conveyor belt.
15: // Note that Init sets a pointer to the conveyor the robot will load.
16: // This version simulates an irregular source of parts by adding a
17: // random number of parts to the conveyor when AddItems() is called.
18: class LoadingRobot
19: {
20:     public:
21:         void Init( Conveyor* WhichConveyor );
22:         void AddItems();
23:     private:
24:         Conveyor* _Conveyor;
25: };
26:
27:
28: #endif
```

```cpp
// A combinatorial logic circuit simulator
//
// The simulated circuit has logic gates and wires to connect them. Test
// signals are generated to test the circuit's output, which is printed
// to screen.
//
// Only NAND gates are supported, and the circuit topology is hard-coded.
//
// This code is functionally complete: it is capable of simulating any non-
// recurrent combinatorial logic function. The provided example demonstrates
// a 2-input, 2-gate, 1-output logic function.
//
// Parts of the code are well-designed and well-written, but some rules of
// object-oriented design are broken, in particular the contents of main are
// not well encapsulated in objects. There are also examples of poor coding
// style including a lack of comments throughout.
//


//--Includes-----------------------------------------------------------------
#include <iostream>
#include <vector>

using namespace std;

//--Consts and enums---------------------------------------------------------
const int InputsPerGate = 2; // number of inputs per nand gate
const int MaxFanout = 2;     // maximum fanout: max gate inputs that one gate output can drive

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

//---Forward Declarations-----------------------------------------------------
class CNandGate; // forward declaration

//---CWire Interface----------------------------------------------------------
// 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();
    void Init();

    // AddOutputConnection adds to the list of outputs that this wire drives
    // It accepts as parameters the nand gate whose input should be driven
    // and the index specifying which of that gate's inputs should be driven.
    void AddOutputConnection(CNandGate *apGateToDrive, int aGateInputToDrive);

    // DriveLevel drives the wire's value, so that each of its connected outputs
    // get set to the corresponding level
    void DriveLevel(eLogicLevel aNewLevel);

private:
    int mNumOutputConnections;            // how many outputs are connected
    CNandGate *mpGatesToDrive[MaxFanout]; // list of connected gates
    int mGateInputIndices[MaxFanout];     // list of input to drive in each gate
};

//---CNandGate Interface------------------------------------------------------
// CNandGate is used to represent NAND gates in this simulation
class CNandGate
{
public:
    // Constructors
    CNandGate();
    void Init();

    // ConnectOutput connects this gate's output to wire {apOutputConnection}
    // That wire will be driven by the gate's output level
    void ConnectOutput(CWire *apOutputConnection);

    // DriveInput drives the specified input of this gate {aInputIndex}
    // with the specified level {aNewLevel}
    void DriveInput(int aInputIndex, eLogicLevel aNewLevel);
```

```
 84:
 85:     // GetOutputState returns the current output level of this gate
 86:     eLogicLevel GetOutputState();
 87:
 88: private:
 89:     void ComputeOutput();
 90:
 91:     eLogicLevel mInputs[InputsPerGate];
 92:     eLogicLevel mOutputValue;
 93:     CWire *mpOutputConnection;
 94: };
 95:
 96: //---CNandCircuit Interface-------------------------------------------------------
 97: // CNandGate is used to represent a circuit made of wires and NAND gates.
 98: // It is initialized with amounts of each respective component.
 99: // Methods allow for the gates and wires to be connected to simulate various circuits,
100: // And the circuit can be tested by driving input wires, then getting gate output levels.
101: class CNandCircuit
102: {
103: public:
104:     // Constructors
105:     // Circuit will have total {gates} gates and {wires} wires
106:     CNandCircuit(int gates, int wires);
107:     void Init(int gates, int wires);
108:
109:     // ConnectWireOutput connects wire number {wire}'s output to input {input} of gate number {gate}
110:     void ConnectWireOutput(int wire, int gate, int input);
111:
112:     // ConnectGateOutput connects gate number {gate}'s output to wire number {wire}
113:     void ConnectGateOutput(int gate, int wire);
114:
115:     // DriveLevel drives wire number {wire} with level {aNewLevel}
116:     void DriveLevel(int wire, eLogicLevel aNewLevel);
117:
118:     // GetOutputState returns the output level of gate number {gate}
119:     eLogicLevel GetOutputState(int gate);
120:
121: private:
122:     int NumNandGates; // number of nand gates
123:     int NumWires;     // number of wires
124:
125:     vector<CNandGate> MyGates; // vector of nand gates
126:     vector<CWire> MyWires;     // vector of wires
127: };
128:
129: //---Or function------------------------------------------------------------
130: // Returns the or result of two input levels.
131: // Builds and simulates an or gate using CNandCircuit.
132: // Very overengineered.
133: eLogicLevel COrGetOutputState(eLogicLevel inputA, eLogicLevel inputB);
134:
135: //---main-------------------------------------------------------------------
136: int main()
137: {
138:     // Test each of the possible input states
139:     std::cout << "Testing input 0 0 or result: " << COrGetOutputState(LOGIC_LOW, LOGIC_LOW) << std::endl;
140:
141:     std::cout << "Testing input 0 1 or result: " << COrGetOutputState(LOGIC_LOW, LOGIC_HIGH) << std::endl;
142:
143:     std::cout << "Testing input 1 0 or result: " << COrGetOutputState(LOGIC_HIGH, LOGIC_LOW) << std::endl;
144:
145:     std::cout << "Testing input 1 1 or result: " << COrGetOutputState(LOGIC_HIGH, LOGIC_HIGH) << std::endl;
146:
147:     return 0;
148: }
149:
150: //---CWire Implementation---------------------------------------------------
151: CWire::CWire()
152: {
153:     Init();
154: }
155:
156: void CWire::Init()
157: {
158:     mNumOutputConnections = 0;
159: }
160:
161: void CWire::AddOutputConnection(CNandGate *apGateToDrive, int aGateInputToDrive)
162: {
163:     mpGatesToDrive[mNumOutputConnections] = apGateToDrive;
164:     mGateInputIndices[mNumOutputConnections] = aGateInputToDrive;
165:     ++mNumOutputConnections;
166: }
```

```
167:
168: void CWire::DriveLevel(eLogicLevel aNewLevel)
169: {
170:     for (int i = 0; i < mNumOutputConnections; ++i)
171:         mpGatesToDrive[i]->DriveInput(mGateInputIndices[i], aNewLevel);
172: }
173:
174: //---CNandGate Implementation--------------------------------------------------
175: CNandGate::CNandGate()
176: {
177:     Init();
178: }
179:
180: void CNandGate::Init()
181: {
182:     mInputs[0] = mInputs[1] = LOGIC_UNDEFINED;
183:     mpOutputConnection = NULL;
184:     ComputeOutput();
185: }
186:
187: void CNandGate::ConnectOutput(CWire *apOutputConnection)
188: {
189:     mpOutputConnection = apOutputConnection;
190: }
191:
192: void CNandGate::DriveInput(int aInputIndex, eLogicLevel aNewLevel)
193: {
194:     mInputs[aInputIndex] = aNewLevel;
195:     ComputeOutput();
196: }
197:
198: eLogicLevel CNandGate::GetOutputState()
199: {
200:     return mOutputValue;
201: }
202:
203: //---CNandCircuit Implementation----------------------------------------------
204: CNandCircuit::CNandCircuit(int gates, int wires)
205: {
206:     Init(gates, wires);
207: }
208:
209: void CNandCircuit::Init(int gates, int wires)
210: {
211:     NumNandGates = gates; // number of nand gates
212:     NumWires = wires;     // number of wires
213:
214:     for (int i = 0; i < NumNandGates; i++)
215:         MyGates.push_back(CNandGate());
216:     for (int i = 0; i < NumWires; i++)
217:         MyWires.push_back(CWire());
218: }
219:
220: void CNandCircuit::ConnectWireOutput(int wire, int gate, int input)
221: {
222:     MyWires[wire].AddOutputConnection(&MyGates[gate], input);
223: }
224:
225: void CNandCircuit::ConnectGateOutput(int gate, int wire)
226: {
227:     MyGates[gate].ConnectOutput(&MyWires[wire]);
228: }
229:
230: void CNandCircuit::DriveLevel(int wire, eLogicLevel aNewLevel)
231: {
232:     MyWires[wire].DriveLevel(aNewLevel);
233: }
234:
235: eLogicLevel CNandCircuit::GetOutputState(int gate)
236: {
237:     return MyGates[gate].GetOutputState();
238: }
239:
240: void CNandGate::ComputeOutput()
241: {
242:     eLogicLevel NewVal = LOGIC_HIGH;
243:     if (mInputs[0] == LOGIC_UNDEFINED || mInputs[1] == LOGIC_UNDEFINED)
244:         NewVal = LOGIC_UNDEFINED;
245:     else if (mInputs[0] == LOGIC_HIGH && mInputs[1] == LOGIC_HIGH)
246:         NewVal = LOGIC_LOW;
247:     mOutputValue = NewVal;
248:
249:     if (mpOutputConnection != NULL)
```

```
250:         mpOutputConnection->DriveLevel(mOutputValue);
251: }
252:
253: //---Or function implementation-----------------------------------------------------------
254: eLogicLevel COrGetOutputState(eLogicLevel inputA, eLogicLevel inputB)
255: {
256:     CNandCircuit circuit(3, 4);
257:
258:     // Wire 0 to gate 0, inputs 0, 1
259:     circuit.ConnectWireOutput(0, 0, 0);
260:     circuit.ConnectWireOutput(0, 0, 1);
261:     // Gate 0 to wire 2
262:     circuit.ConnectGateOutput(0, 2);
263:
264:     // Wire 1 to gate 1, inputs 0, 1
265:     circuit.ConnectWireOutput(1, 1, 0);
266:     circuit.ConnectWireOutput(1, 1, 1);
267:     // Gate 1 to wire 3
268:     circuit.ConnectGateOutput(1, 3);
269:
270:     // Wire 2 to gate 2 input 0
271:     circuit.ConnectWireOutput(2, 2, 0);
272:     // Wire 3 to gate 2 input 1
273:     circuit.ConnectWireOutput(3, 2, 1);
274:
275:     // Drive wires 0, 1, get gate 2 output
276:     circuit.DriveLevel(0, inputA);
277:     circuit.DriveLevel(1, inputB);
278:     return circuit.GetOutputState(2);
279: }
```