Abstract

Molecular modeling provides a means of expressing complex ideas in chemistry in an intuitive, concrete manner. Computers are already heavily used in the field as a means of storing enormous databases of information about molecules, chemical properties, spectral data, and reactions, and as a means of visualizing complex molecules in three dimensions. Presented here is the beginning of a project to pull together currently separate technologies and integrate them into a powerful, functional, yet simple to use interface to enhance the capabilities of both physical molecular models and the vast software resources currently available. The design process for this construction kit reveals a number of the issues that would need to be addressed before this kit could be realized in a truly meaningful form, especially with respect to the implementation of the physical modeling set. The current form of the kit sidesteps the hardware issue, choosing instead to provide input that models the way the hardware would likely communicate with the software by using a barcode labeling system and a barcode scanner. The final software design establishes base classes for storing information about atoms and molecules as well as providing a few rudimentary manipulation functions and a comparison routine for matching a model against a small database of known molecules. The majority of the software needed to provide additional functionality such as recognition of isomorphic molecules, 3D visualization and searching of large online databases by structure is already available, and would need only to be integrated to form an intuitive yet powerful interface between the physical models and the wealth of information available online. This would be useful at all levels of chemistry, ranging from a guided modeling tutorial as a hands-on introduction to the topic of chemistry to a 3D sketching program for visualizing complex molecules in a research laboratory setting.

Design

Design Process
The motivation for this project stems partially from the repeated in-class discussion of the problems associated with building a smart molecular modeling set, and partially from the composition of the group. Between the three group members, we will shortly be able to account for chemistry, computer science and electrical and computer engineering degrees, putting us in a unique position to address issues relating to the needs of the intended user as well as both hardware and software requirements for this project. The
largest difficulty with this project was the lack of time. Implementing the relatively simple interface used in this project required a significant amount of coding to establish a reliable software base, and sidestepped the issue of a hardware implementation by emulating the ideal input with a barcode labeling/scanning mechanism.

Our first design phase focused on establishing a proof of concept hardware system. Early suggestions for this design included using crickets embedded in atoms directly and communicating with IR over fiber optics, or receiving input via a cricket sensor port from a multiplexer connected to several atoms. These cricket-based implementations were quickly scraped, due to the severe limitations of crickets with respect to this project. Specifically, the analog to digital conversions used by the cricket sensor ports are too coarse to use as a reliable, controllable means of communication, while IR communications are limited to one direction, or possibly two with the use of fiber optics. A non-trivial organic molecule would require no less than four connections per atom. Crickets also contain their own power source, making them bulky. A cricket in each atom would make the model too large for easy use. Crickets also posses significantly more computational power than would be required in this case.

Having established that the available hardware was not a feasible solution to the problems presented by this project, we briefly experimented with the idea of implementing custom hardware. While this would have allowed us the luxury of a hardware implementation that could have addressed all the shortcomings of the crickets, the time available for the project prohibited us from realistically attempting this solution. Nat suggested that a printed circuit board would be a good hardware solution for the size considerations, computing and power requirements. However, he estimated that this would require no less than one week to fabricate, and could not be designed, tested and fabricated in the four weeks available.

Materials
Our final iteration of the project included a barcode scanner, some of James’ old molecular models, and printed barcodes tied to individual atoms with thread. The rest of the project was implemented in software.

Current Hardware
The final version of the hardware used barcodes attached to the molecular modeling set. A barcode scanner was used to read the labels to standard input. This was done to simulate the input we would expect from the ideal hardware. As advertised, the barcode scanner outputs directly to standard input. This allowed us to integrate the scanner easily into the project, despite only acquiring it four days before the presentation.

Ideal Hardware
Given a much longer time to work on this project, the hardware requirements could have been addressed. The ideal hardware would consist of a microcontroller that would allow at least four independent input/output ports. Nat also suggested an onboard clock to reduce the size, minimizing the required number of parts external to the printed circuit board to increase robustness, and minimizing the time between plugging an atom into an
external power source and actually reading a signal at the terminal. The hardware would need almost no computational power and minimal storage at each atom. Atom identities and connection information would be passed to the computer to be represented and manipulated in software. The full development cycle needed to get a solid, reliable hardware implementation would take enough time to make this an appropriate project for a graduate student.

**Current Software**

The final software package consists of a small, cohesive suite of classes implemented in Java. Java was used to facilitate rapid, high-level, object-oriented development of the interface, as well as a means to ensure code portability. Specifically, it was not known what systems the barcode scanner would easily interface with, so the code had to be able to run on a machine that could support the scanner.

The software consists of four classes. The first, Atom, is strictly a data container, storing information about its type and unique numbering, as well as references to the other Atoms it is bonded to. The Molecule class provides dynamic storage for Atoms and basic manipulation functions such as 2D printing of the current molecule. The printing function implemented here is quite primitive, and works by naively assigning adjacent atoms to the right, left, bottom and finally top of the current atom as needed. This occasionally results in overlapping atoms in the output. DBFileWrapper reads a structure from a flat file in a ‘database’ of known molecules. TestStructure provides a text interface for handling user input, hardware events, and detection of completed structures. Currently, TestStructure does not recognize isomorphic representations of the same molecule.

**Ideal Software**

There are several major improvements that could be made to greatly increase the functionality of the software package. The first would be to provide a graph isomorphism detection algorithm that can produce a unique representation of a molecule independent of the orientation of the input graph. The nauty algorithm by Brendan McKay, of the Australian National University, produces the required ‘canonical labeling’ at reasonable speeds for small and moderate graph sizes. The C code is available at http://cs.anu.edu.au/~bdm/nauty/.

Many of the problems with overlap in the 2D structures used as output in this project would be solved by using a graphical 3D viewer. The Java applet shown on http://www.netbrain.com/~brain/molecule/ gives a good example of 3D molecule visualization, and is available as Java source code. Note that the input used by this applet is a .pdb file, which specifies what an atom is and were it is located in 3D. This could easily be modified to calculate relative atom positions. This would significantly decrease the number of overlaps by avoiding the projection of inherently 3D data into a 2D output format. A graphical interface, such as the applet shown here, allows manipulation of the object, allowing it to be rotated for ease of use. This would significantly increase the interactive appeal of the design. The MDL Chime plugin provides similar functionality.
with the advantage of being a supported commercial product, but with the disadvantage of not including source code. The plugin can be downloaded from http://www.mdl.com.

The addition of a true database of organic molecules with the option to search by structure would aid in rapid identification of the molecule, allowing chemical properties, research literature, and spectra to be keyed to the hardware structure. Currently, several organic chemistry databases are online and support a search by structure feature, such as Beilstein Crossfire, another commercial product from MDL, available at http://www.mdl.com.

The interesting thing about all of the software improvements suggested here is that these are problems that have already been addressed by the chemistry and computer science communities. Each has been solved separately of the others. As yet, there is no package that integrates these components to form a cohesive software package.

**Evaluation / Education**

In its current form, our modeling set provides a good summary of all the difficult problems that would need to be addressed in order to make a robust, usable implementation of a smart molecular modeling set. In its final version, this construction kit would provide a useful, educational experience for a wide variety of users. It would find application among organic chemistry students as a method of exploring properties of a molecule, effects of conformation changes, and effects of stepwise reaction sequences. A simplified version could be used as a hands-on introduction to chemistry, perhaps in the form of a computer guided tutorial providing interactive feedback, tutorials on building simple molecules, challenge problems, and a discussion of the interesting features of some pre-constructed models. This kit might also be useful in a research lab as a means of ‘sketching’ a molecule of interest in 3D, avoiding awkward 2D representations of complex molecules with inherent structure. Watson and Crick’s use of rudimentary molecular modeling to try and better understand the structure of DNA provides a perfect example of what an important role 3D sketching plays in understanding complex molecules.

The current form of our kit is not particularly intuitive. It requires the user to bear the burden of ensuring correct connections and scanning the atoms in an order that reflects the physical connections. This was done to simulate the sort of input that the software would expect from the ideal implementation of the hardware. There is currently no way to remove an atom if you make a mistake, the program exits if the user makes an error, and the limitations of a 2D viewer prohibit the construction and visualization of more than the simplest of organic molecules.

In final form, the kit would be completely intuitive to anyone with the knowledge of a first year chemistry class, and could be made highly accessible even without this knowledge by providing the software tutorial discussed above. This could make chemistry a far more accessible science, at least at the introductory level.
This kit is simply an extension of the molecular modeling kits already available on the market. The physical pieces should be very similar to those provided with any kit, making the hardware easy to recognize and use. On the software level, the functionality we envision is mostly available as off-the-shelf components. Currently, these are separate and not designed to interact with each other. By assembling these components into a cohesive whole, the software could provide an interface that is intuitive and flexible enough to meet the needs of a wide range of users with varying degrees of chemistry knowledge.
Appendix 1 – Images

Image 1 – Methanol, tagged with barcodes

Image 2 – Individual tagged atoms and barcode scanner
Appendix 2 – Code & Test Cases

Makefile

OBJS = Molecule.class Atom.class $(PROG).class
PROG = TestStructure
OPTS = -classpath .

all: $(OBJS)
run: $(PROG).class
  java $(OPTS) $(PROG)
%.class: %.java
  javac $(OPTS) $<
clean:
  rm -f *.class

TestStructure.java (Main Program)

import java.util.Vector;
import java.io.*;
import java.util.Enumeration;

public class TestStructure
{
  private static Molecule myMol = new Molecule();
  private static Vector testMolecules = new Vector();
  private static Vector dbFiles = new Vector();

  public static void main(String[] args)
  {
    boolean startOver = true; // Variable to decide whether or not
    // to start the molecule over
    Atom newAtom; // New Atom to add

    try
    {
      // Read in "database" files
      readDatabase();

      // Wait for scanner input
      while(true)
      {
        if(startOver) // Make a new molecule and read in the first atom
        {
          startOver = false;

          myMol.clearMolecule();

          // Get the first atom
          System.out.println("\nBuild a molecule!\n");
          System.out.println("When finished, please scan or type 'DONE'\n");
          System.out.print("First Atom: ");
newAtom = getAtomFromScanner();

if(newAtom == null)
{
    System.exit(-1);
}

// And add it to the molecule
myMol.add(newAtom);

} // End of if(startOver)

System.out.print("New Atom:  ");
newAtom = getAtomFromScanner();

if(newAtom == null)
    break;

System.out.print("Atom to attach to:  ");
Atom existingAtom = getAtomFromScanner();

if(existingAtom == null)
{
    System.out.println("Error reading existing atom
Exiting");
    System.exit(-1);
}

// Add the new atom on to the molecule
myMol.add(newAtom, myMol.getAtom(existingAtom));

// Print out the current structure
String currentStructure = myMol.structure();
System.out.println("Current structure:
" + currentStructure);

// Now, compare the structure of this molecule
// with some known ones.
System.out.println(); // Some space

boolean recognized = false;
Enumeration files = dbFiles.elements();
while(files.hasMoreElements())
{
    DBFileWrapper nextFile = (DBFileWrapper) files.nextElement();
    String targetStructure = nextFile.getStructure();
    // Now, test and see whether we recognize this molecule
    if (targetStructure.equals(currentStructure))
    {
        System.out.println("Molecule recognized as " +
            nextFile.getStructureName());
        recognized = true;
        System.out.println("\nWould you like to start over [Y/n]?
");
        String line = "";
    }
BufferedReader b = new BufferedReader(new InputStreamReader(System.in));

// Read from stdin
try {line = b.readLine();}
catch (Exception e) {e.printStackTrace();}

if( (line.substring(0, 1)).equals("n") )
    startOver = false;
else
    startOver = true;

break;
}

if(!recognized)
    System.out.println("Molecule unrecognized");

System.out.println(); // Some more space

} // End of infinate loop

} // End of main

public static void readDatabase()
{
    // Set up the "database" files
dbFiles.addElement(new DBFileWrapper("water.txt"));
dbFiles.addElement(new DBFileWrapper("methanol.txt"));
dbFiles.addElement(new DBFileWrapper("ethanol.txt"));
dbFiles.addElement(new DBFileWrapper("isopropanol.txt"));

    // Now, read in the files
Enumeration files = dbFiles.elements();
while(files.hasMoreElements())
{
    DBFileWrapper nextFile = (DBFileWrapper) files.nextElement();

    nextFile.readInStructure();

    // Save the contents of this file into our testMolecules vector
testMolecules.addElement(nextFile.getStructure());
}
} // End of readDatabase

// Return null if DONE was scanned
// (or something else we didn't understand)
public static Atom getAtomFromScanner()
// Set up stuff to read from stdin
String line = ";
BufferedReader b = new BufferedReader(new
InputStreamReader(System.in));

// Read from stdin
try
{
    // Read one line
    line = b.readLine();
} catch (Exception e)
{
    e.printStackTrace();
}

// Strip out the Atom info and do some error checking
char type = line.charAt(0);
if(!( (type == 'C') || (type == 'O') || (type == 'H') || (type == 'N') ))
{
    return null;
}

int id;
try
{
    id = Integer.parseInt(line.substring(1));
} catch(NumberFormatException e)
{
    // We got something we didn't understand
    return null;
}

return(new Atom(type, id));

} // End of getAtomFromScanner

---------------------------------------------------------------------
DBFileWrapper.java (Interface for reading flat 'database' files)
---------------------------------------------------------------------
import java.io.*;

public class DBFileWrapper
{
    private String _filename;
    private String _structureName;
    private String _structure;
    private File _file;

    // Constructor
    public DBFileWrapper(String filename)
    {
        _filename = filename;
    }
_structure = "";

// To get the name of the structure, strip off the .txt
// and make the first letter capital
String tmp = _filename.substring(0, _filename.indexOf('.'));
_structureName = (tmp.substring(0, 1)).toUpperCase() +
    tmp.substring(1);

_file = new File(_filename);
}

public void readInStructure()
{
    LineNumberReader in;
    try
    {
        in = new LineNumberReader(new FileReader(_file));

        _structure   = null;
        String line   = null;

        while ((line = in.readLine()) != null)
        {
            if (_structure != null)
            {
                _structure += line;
            }
            else
            {
                _structure = line;
            }
            _structure += "\n";
        }
    } catch (FileNotFoundException e)
    {
        System.out.println("File " + _filename + " not found
Exiting");
        System.exit(-1);
    } catch (IOException e)
    {
        e.printStackTrace();
    }

    // strip off last \n
    _structure = _structure.substring(0, _structure.length()-1);
}

public String getStructure()
{
    if(_structure.equals"")) // Haven't read in structure yet
        this.readInStructure();
    return _structure;
}

public String getStructureName()
public class Molecule
{
    // Declare a vector ("array") of atoms
    public static Vector _atoms = new Vector();
    protected Atom[][] atoms_array;

    // Constructor
    public Molecule()
    {
    }

    // Method to clear out a molecule to start over
    public void clearMolecule()
    {
        _atoms.clear();
    }

    // Add newAtom
    public void add(Atom newAtom)
    {
        // Put the new atom into our molecule list
        _atoms.addElement(newAtom);
    }

    public void add(Atom newAtom, Atom existingAtom)
    {
        // Attach the atoms to each other
        existingAtom.attach(newAtom);
        newAtom.attach(existingAtom);

        // Put the new atom into our molecule list
        _atoms.addElement(newAtom);
    }

    public Atom getAtomAt(int index)
    {
        return (Atom) _atoms.elementAt(index);
    }

    // If a copy of Atom a is in the Vector, return it.
    // Otherwise, exit (for now)
    public Atom getAtom(Atom desiredAtom)
    {
        int index = _atoms.indexOf(desiredAtom);
        if(index == -1)
        {
            System.out.println("Element not found");
            System.exit(-1);
        }
        return (Atom) _atoms.elementAt(index);
    }
}
return(null);
}
else
return this.getAtomAt(index);
}

public int getNumFreeSpaceAround(int x, int y)
{
    int num = 0;
    if(atoms_array[x+2][y].getType() == '0')
        num++;
    if(atoms_array[x-2][y].getType() == '0')
        num++;
    if(atoms_array[x][y-2].getType() == '0')
        num++;
    if(atoms_array[x][y+2].getType() == '0')
        num++;
    return num;
}

public void commitNextAtom(Atom commitAtom, int x, int y, int from)
{
atoms_array[x][y] = commitAtom;
if(commitAtom.getNumBond() > 1)
{
    if(getNumFreeSpaceAround(x,y) < (commitAtom.getNumBond() - 1))
    {
        System.out.println("There is not currently enough space to
print this molecule");
        System.exit(-1);
    }
    switch(from)
    {
        case 4:
            switch(commitAtom.getNumBond())
            {
                case 4:
                    try
                    {
                        commitNextAtom(atoms_array[x][y].getNextBond(3), x, y-2, 1);
                        atoms_array[x][y-1] = new Atom('-');
                    }
                    catch(Exception e){}
                case 3:
                    try
                    {
                        commitNextAtom(atoms_array[x][y].getNextBond(2), x, y+2, 2);
                        atoms_array[x][y+1] = new Atom('-');
                    }
                    catch(Exception e){}
                case 2:
                    try
                    {
                        commitNextAtom(atoms_array[x][y].getNextBond(1), x, y, 2);
                        atoms_array[x][y-1] = new Atom('-');
                    }
                    catch(Exception e){}
                case 1:
                    try
                    {
                        commitNextAtom(atoms_array[x][y].getNextBond(0), x, y, 1);
                        atoms_array[x][y] = new Atom('-');
                    }
                    catch(Exception e){}
                default:
                    System.out.println("Invalid bond number");
                    System.exit(-1);
            }
    }
}

catch(Exception e){}
case 2:
    try {
        commitNextAtom(atoms_array[x][y].getNextBond(1), x+2,
                       y, 4);
        atoms_array[x+1][y] = new Atom('|');
    } catch(Exception e){}
    break;

case 3:
    switch(commitAtom.getNumBond())
    {
    case 4:
        try {
            commitNextAtom(atoms_array[x][y].getNextBond(3), x, y-2,
                           1);
            atoms_array[x][y-1] = new Atom('-');
        } catch(Exception e){}
    case 3:
        try {
            commitNextAtom(atoms_array[x][y].getNextBond(2), x,
                           y+2, 2);
            atoms_array[x][y+1] = new Atom('-');
        } catch(Exception e){}
    case 2:
        try {
            commitNextAtom(atoms_array[x][y].getNextBond(1), x-2,
                           y, 3);
            atoms_array[x-1][y] = new Atom('|');
        } catch(Exception e){}
    break;
    case 2:
        switch(commitAtom.getNumBond())
        {
        case 4:
            try {
                commitNextAtom(atoms_array[x][y].getNextBond(3), x-2,
                               y, 3);
                atoms_array[x-1][y] = new Atom('|');
            } catch(Exception e){}
        case 3:
            try {
            }
commitNextAtom(\text{atoms\_array}[x][y].\text{getNextBond}(2), x+2, y, 4);
\text{atoms\_array}[x+1][y] = \text{new \text{Atom}}('|');
}
\text{catch(Exception e)}{}
case 2:
\text{try}
{
commitNextAtom(\text{atoms\_array}[x][y].\text{getNextBond}(1), x, y+2, 2);
\text{atoms\_array}[x][y+1] = \text{new \text{Atom}}('-');
}
\text{catch(Exception e)}{}
break;

case 1:
\text{switch(\text{commitAtom}\text{.getNumBond()})}
{
\text{case 4:}
\text{try}
{
commitNextAtom(\text{atoms\_array}[x][y].\text{getNextBond}(3), x-2, y, 3);
\text{atoms\_array}[x-1][y] = \text{new \text{Atom}}('|');
}
\text{catch(Exception e)}{}
\text{case 3:}
\text{try}
{
commitNextAtom(\text{atoms\_array}[x][y].\text{getNextBond}(2), x+2, y, 4);
\text{atoms\_array}[x+1][y] = \text{new \text{Atom}}('|');
}
\text{catch(Exception e)}{}
\text{case 2:}
\text{try}
{
commitNextAtom(\text{atoms\_array}[x][y].\text{getNextBond}(1), x, y+2, 1);
\text{atoms\_array}[x][y+1] = \text{new \text{Atom}}('-');
}
\text{catch(Exception e)}{}
break;
}
\text{// End of switch(from)}
\text{// End of if(\text{commitAtom}\text{.getNumBond()} > 1)}
\text{// End of commitNextAtom}

\text{public String structure()}
{
\text{String structure} = "";
\text{Enumeration atoms = }_\text{atoms}.\text{elements();}
\text{int atoms\_count = }_\text{atoms}.\text{size();}
int first_atom = 1, atom_number = 0;
int i,j;

atoms_array = new Atom[atoms_count*4][atoms_count*4];
for(i=0;i<atoms_count*4;i++)
    for(j=0;j<atoms_count*4;j++)
        atoms_array[i][j] = new Atom('0');

atoms_array[atoms_count*2][atoms_count*2] = (Atom) atoms.nextElement();

switch(atoms_array[atoms_count*2][atoms_count*2].getNumBond())
{
    case 4:
        // NOTE: getNextBond() calls Vector.elementAt()
        // which throws ArrayIndexOutOfBoundsException
        try
        {
            commitNextAtom(atoms_array[atoms_count*2][atoms_count*2].getNextBond(3)
                , (atoms_count*2)-2, atoms_count*2, 3);
            atoms_array[(atoms_count*2)-1][atoms_count*2] = new Atom('|'); // Up
        }
        catch(Exception e){}
    case 3:
        try
        {
            commitNextAtom(atoms_array[atoms_count*2][atoms_count*2].getNextBond(2)
                , (atoms_count*2)+2, atoms_count*2, 4);
            atoms_array[(atoms_count*2)+1][atoms_count*2] = new Atom('|'); // Down
        }
        catch(Exception e){}
    case 2:
        try
        {
            commitNextAtom(atoms_array[atoms_count*2][atoms_count*2].getNextBond(1)
                , atoms_count*2, (atoms_count*2)-2, 1);
            atoms_array[atoms_count*2][(atoms_count*2)-1] = new Atom('-'); // Left
        }
        catch(Exception e){}
    case 1:
        try
        {
            commitNextAtom(atoms_array[atoms_count*2][atoms_count*2].getNextBond(0)
                , atoms_count*2, (atoms_count*2)+2, 2);
            atoms_array[atoms_count*2][(atoms_count*2)+1] = new Atom('-'); // Right
        }
        catch(Exception e){}
}
int first_column = 0;

for(j=0;j<(atoms_count*4);j++)
{
    for(i=0;i<(atoms_count*4);i++)
    {
        Atom next = atoms_array[i][j];
        if(next.getType()!='0')
        {
            first_column = j;
            break;
        }
    }
    if(first_column != 0)
    break;
}

for(i=0;i<(atoms_count*4);i++)
{
    int row_length = 0;
    int since_last_atom = 0;
    for(j=first_column;j<(atoms_count*4);j++)
    {
        if(atoms_array[i][j].getType()!='0')
        {
            row_length += (since_last_atom+1);
            since_last_atom = 0;
        }
        else
        {
            since_last_atom++;
        }
    }
    for(j=first_column;j<(atoms_count*4);j++)
    {
        atoms_array[i][j-first_column] = atoms_array[i][j];
    }
    for(j=row_length+1;j<(atoms_count*4);j++)
    {
        atoms_array[i][j] = new Atom('0');
    }
}

for(i=0;i<(atoms_count*4);i++)
{
    int empty_row = 1;
    int empty_back = 0;
    for(j=0;j<(atoms_count*4);j++)
    {
        Atom next = atoms_array[i][j];
        if(next.getType()!='0')
        {
            empty_back = 0;
            empty_row = 0;
            structure += next.getType();
        }
    }
else
{
   structure += " ";
   empty_back++;
}
}
if(empty_row == 0)
{
   structure = structure.substring(0, structure.length() -
   (empty_back));
   structure += '\n';
} else
   structure = structure.substring(0, structure.length() -
   (atoms_count*4));

   // Strip off last "\n"
   structure = structure.substring(0, structure.length()-1);

   return structure;
}
} // End of structure()
break;
case 'O':
    _numBonds = 2;
    break;
case 'N':
    _numBonds = 3;
    break;
case 'C':
    _numBonds = 4;
    break;
}
}
public char getType()
{
    return _type;
}

public int getID()
{
    return _id;
}

public int getNumBond()
{
    return _numBonds;
}

public void attach(Atom a)
{
    // Check for too many bonds
    if( !_bonds.isEmpty() && (_bonds.size() == _numBonds) )
    {
        System.out.println("No bonds available on this atom");
        System.exit(-1);
    }
    // Attach it
    _bonds.addElement(a);
}

public Atom getNextBond(int index)
{
    return (Atom) _bonds.elementAt(index);
}

    // Overridden method from Object class
    public boolean equals(Object obj)
    {
        Atom a = (Atom) obj;

        return( (this.getType() == a.getType()) && (this.getID() == a.getID()) );
    }
}
methanol.txt (Example ‘database’ file)

H
| H-C-O-H
| H

C0
O0
C0
H0
O0
H1
C0
H2
C0
H3
C0
n
DONE