Assessed Exercise

This exercise is based on the `sac16` Java package, which provides a 16 bit single key encryption system. There are three parts to the exercise. **Send me an email with the subject SAC** and I will then email you your 3 individual cipher texts and 2 known plain texts.

Once you have finished your exercise, send me an email with a zip file containing a report and your four programs, called `KPT.java`, `CTO.java`, `TMT1.java` and `TMT2.java`. Your zip file should have the form `lastname_firstname.zip` - mine would be called `poet_ron.zip`.

Deadline Mon 24th February at 4.30 p.m. This exercise is worth 20% of your mark for this module.

Known Plain Text Attack

You will be given a chunk of cipher text, together with the plaintext corresponding to the first block. Write a program to perform a brute force search of the key space to find the key. Use this key to decode the rest of the message.

Your report should contain the key you found and the decoded message.

Cipher Text Only Attack

You will be given another chunk of cipher text, as above, but no plaintext. You will need to perform a brute force attack as before, knowing that the plaintext is an English language message. You will also need to perform an experiment to find out how many cipher text blocks were needed to decode the message unambiguously.

Your report should contain the key you found and the decoded message. You should also include a theoretical calculation of the number of blocks of cipher text needed before unambiguous decoding is possible, together with the actual number of blocks needed for your particular message, as found by your experiment.

Time Memory Tradeoff

You will be given the first plaintext block, which you should use to construct a time memory tradeoff table. This table should be written to a file, and the simplest way of doing this is to write the entries as two integers per line.

Some time later you will be given the full cipher text, and your second program should read in the table entries from the file, construct the table, obtain the first cipher text block, discover the key and decode the rest of the cipher text.

Since the time-memory tradeoff is a probabilistic attack, it is possible that the key I have used is not in your table. In this case, run the table generating program again with different random values and try again.

Your report should contain the key you have found and the deciphered text.

The sac16 Code

The code is written in Java, but you should not have much trouble writing in this language since I will provide sample code. My code uses the `FormatIO` package for simple input and output. All code and the `FormatIO` manual are in the zip file on Moodle. You do not have to use FormatIO.

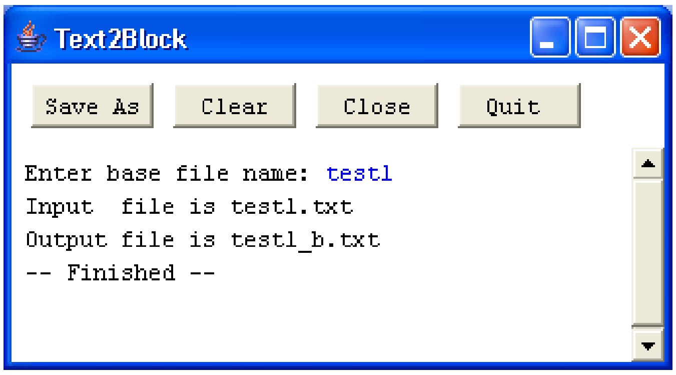
Blocking

The code converts from text to 16 bit blocks and vice versa. 16 bit block files are written as 16 bit integers, in hexadecimal notation, one per line.

There are two program, `text2block` and `block2text` to handle the conversion. Each program converts its standard input to its standard output.

Text2Block

This program takes ascii text as an input file, converts it to 16 bit blocks and then outputs the results as a series of 16 bit hexadecimal integers, one per line to an output file. If the initial text has an odd number of characters then a character with ascii code 0 as added to the end. Here is a screenshot.



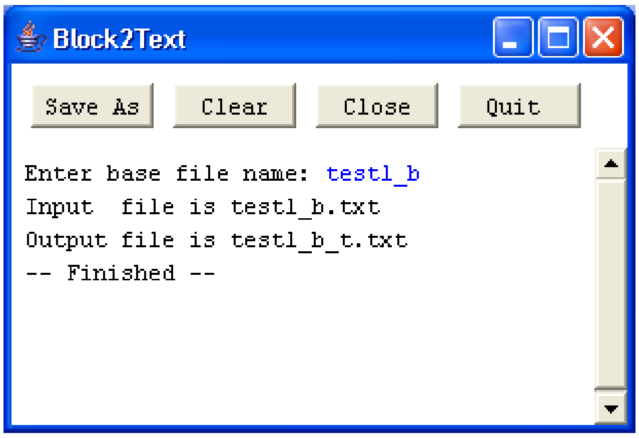
```
Enter base file name: test1
Input  file is test1.txt
Output file is test1_b.txt
-- Finished --
```

Here is the code.

```
/**
 *  Text2Block converts a text file to a block file.
 *  If the text has an odd number of characters then
 *  an extra ascii 0 character is added.
 *  (c) Ron Poet
 */

import FormatIO.*;

public class Text2Block
{
      private     static      FileIn      fin;
      private     static      FileOut     fout;
public      static      void  main(String[] arg)
{
            // create a Console for IO
      Console     con = new Console("Text2Block");

            // get file names
      con.print("Enter base file name: ");
      String      name = con.readWord();
      con.println("Input  file is " + name + ".txt");
      con.println("Output file is " + name + "_b.txt");

            // open files
      fin = new FileIn(name + ".txt");
      fout = new FileOut(name + "_b.txt");

            // read chars and output blocks
      int   count = 0;
      char  c0 = 0;
      try
      {
            for (;;)
            {
                  char  c1 = fin.getChar();
                  count++;
                  if (count % 2 == 0)     // two chars for full block
                        out2(c0, c1);
                  c0 = c1;    // remember this one
            }
      }
      catch (EofX x)
      {
            if (count % 2 == 1)     // odd number, pad with 0
                  out2(c0, (char)0);
      }
      fout.close();
      con.println("-- Finished --");
}

      public      static      void  out2(char c0, char c1)
      {
            String      out = String.format("0x%02x%02x",
                  (int) c0, (int) c1);
            fout.println(out);
      }
}
```

Block2Text

This reverses `Text2Block`, removing any added ascii 0 character. Here is a screenshot:



```
Enter base file name: testl_b
Input  file is testl_b.txt
Output file is testl_b_t.txt
-- Finished --
```

Here is the code.

```
/**
 *  Block2Text converts a block file to a text file.
 *  Any extra ascii 0 character is discarded.
 *  (c) Ron Poet
 */

import FormatIO.*;

public class Block2Text
{
      private     static      FileIn      fin;
      private     static      FileOut     fout;

public      static      void  main(String[] arg)
{
            // create a Console for IO
      Console     con = new Console("Block2Text");

            // get file names
      con.print("Enter base file name: ");
      String      name = con.readWord();
      con.println("Input  file is " + name + ".txt");
      con.println("Output file is " + name + "_t.txt");

            // open files
      fin = new FileIn(name + ".txt");
      fout = new FileOut(name + "_t.txt");

            // read blocks and output chars
      try
      {
            for (;;)
            {
                  String      s = fin.readWord();
                  int   i = Hex16.convert(s);
                  int   c0 = i / 256;
```

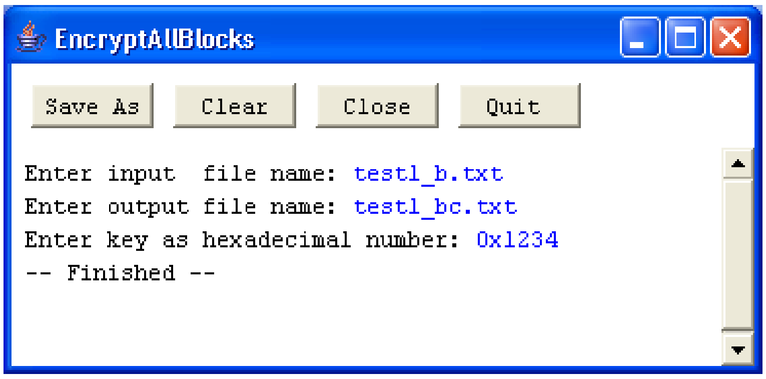
```
                int   c1 = i % 256;
                fout.print((char)c0);
                if (c1 != 0)
                      fout.print((char)c1);
          }
      }
      catch (EofX x)
      {
      }
      fout.close();
      con.println("-- Finished --");
}

}
```

It uses a helper class Hex16 to convert from a hex string to an int.

```
/**
 * Converts a 16 bit 4-hex-digit string to an int.
 * @author ron
 *
 */
public class Hex16
{
      public      static      int   convert(String s)
      {
            int   i0 = hex2int(s.charAt(2));
            int   i1 = hex2int(s.charAt(3));
            int   i2 = hex2int(s.charAt(4));
            int   i3 = hex2int(s.charAt(5));
            return i3 + 16 * (i2 + 16 * (i1 + 16 * i0));
      }

      private     static      int   hex2int(char c)
      {
            if (c >= '0' && c <= '9')
                  return (int)(c - '0');
            else if (c >= 'a' && c <= 'f')
                  return (int) (c - 'a') + 10;
            else if (c >= 'A' && c <= 'F')
                  return (int) (c - 'A') + 10;
            else
                  return 0;
      }
}
```

Encryption and Decryption

There are two programs to handle encryption and decryption of blocked data. Each program takes a 16 bit key (written in hex notation) and converts an input file to an output file. Here is a screen shot



```
Enter input  file name: test1_b.txt
Enter output file name: test1_bc.txt
Enter key as hexadecimal number: 0x1234
-- Finished --
```

EncryptAllBlocks

Here is the code:

```
/**
 * Encrypts a file containing blocks as hex numbers
 * Output is another file containing blocks
 * (c) Ron Poet
 */

import FormatIO.Console;
import FormatIO.EofX;
import FormatIO.FileIn;
import FormatIO.FileOut;

public class EncryptAllBlocks
{
      public      static      void  main(String[] arg)
      {
            // create a Console for IO
            Console     con = new Console("EncryptAllBlocks");

                  // get file names
            con.print("Enter input  file name: ");
            String      name_in = con.readWord();
            con.print("Enter output file name: ");
            String      name_out = con.readWord();

                  // get key
            con.print("Enter key as hexadecimal number: ");
            String      key_string = con.readWord();
            int   key = Hex16.convert(key_string);

                  // open files
            FileIn      fin   = new FileIn(name_in);
            FileOut fout = new FileOut(name_out);

            // read blocks, encrypt and output blocks
            try
            {
                  for (;;)
```

```
                {
                      String      s = fin.readWord();
                      int   p = Hex16.convert(s);
                      int   c = Coder.encrypt(key, p);
                      String      out = String.format("0x%04x", c);
                      fout.println(out);
                }
          }
          catch (EofX x)
          {
          }
          fout.close();
          con.println("-- Finished --");
    }

}
```

DecryptAllBlocks

Here is the code:

```
/**
 * Decrypts a file containing blocks as hex numbers
 * Output is another file containing blocks
 * (c) Ron Poet
 */

import FormatIO.Console;
import FormatIO.EofX;
import FormatIO.FileIn;
import FormatIO.FileOut;

public class DecryptAllBlocks
{
      public      static      void  main(String[] arg)
      {
            // create a Console for IO
            Console     con = new Console("EncryptAllBlocks");

                  // get file names
            con.print("Enter input  file name: ");
            String      name_in = con.readWord();
            con.print("Enter output file name: ");
            String      name_out = con.readWord();

                  // get key
            con.print("Enter key as hexadecimal number: ");
            String      key_string = con.readWord();
            int   key = Hex16.convert(key_string);

                  // open files
            FileIn      fin  = new FileIn(name_in);
            FileOut fout = new FileOut(name_out);
            // read blocks, encrypt and output blocks
            try
            {
                  for (;;)
                  {
                        String      s = fin.readWord();
                        int   c = Hex16.convert(s);
                        int   p = Coder.decrypt(key, c);
                        String      out = String.format("0x%04x", p);
                        fout.println(out);
                  }
            }
            catch (EofX x)
            {
            }
            fout.close();
            con.println("-- Finished --");
      }

}
```

Encrypting and Decrypting Single Blocks

Both these programs use the Coder class for encryption and decryption. I will not provide a .java file because I don't want you to make a cryptographic attack. The methods are:

```
public    static    int  encrypt(int key, int plain)
public    static    int  decrypt(int key, int cipher)
```

You can see how they are used from the sample programs EncryptAllBlocks and DecryptAllBlocks. You will have to use these methods in your own programs.

Table Class

I have provided a `Table` class for the time-memory tradeoff question. There are two methods: `add` to add two integers (key, data) to the table; and `find` where you provide the key and it either returns the data (if the key were in the table) or -1 of it was not. Here is the code.

```
/**
 * Lookup table based on a HashMap
 * @author ron
 *
 */

import java.util.HashMap;
import java.util.Map;
```

```
public class Table
{
      private     Map   tab;

      public      Table()
      {
            tab = new HashMap();
      }

      public      void  add(int key, int data)
      {
            tab.put(new Integer(key), new Integer(data));
      }

      public      int   find(int key)     // return data or -1
      {
            Integer     keyobj = new Integer(key);
            if (tab.containsKey(keyobj))
            {
                  Integer     io = (Integer) tab.get(keyobj);
                  return io.intValue();
            }
            else
                  return -1;
      }
}
```

Random

The class `java.util.Random` will provide random numbers.