Diploma Programme

Computer science teacher support material

Internal assessment

First examinations 2010

Diploma Programme

Computer science

Internal assessment

Teacher support material

First examinations 2010

International Baccalaureate Organization

Buenos Aires Cardiff Geneva New York Singapore

Diploma Programme

Computer science: internal assessment—teacher support material

First published May 2005
Revised edition published September 2008

International Baccalaureate
Peterson House, Malthouse Avenue, Cardiff Gate
Cardiff, Wales GB CF23 8GL
United Kingdom
Phone: +44 29 2054 7777
Fax: +44 29 2054 7778
Website: http://www.ibo.org

Printed in the United Kingdom by Antony Rowe Ltd, Chippenham, Wiltshire

Contents

Introduction

The new courses for computer science at standard level (SL) and at higher level (HL) are examined for the first time in May 2006.

This teacher support material (TSM) supplements and should be read alongside the *Computer Science* guide (published in April 2004), which contains the philosophy, requirements and regulations for the whole course. Where relevant, extracts from the guide have been reproduced in this TSM for ease of reference. General regulations and procedures relating to internal assessment (IA) have not been reproduced here but can be found in the *Vade Mecum*. This TSM aims to amplify certain aspects of the syllabus content, to clarify some aspects of the program dossier and to exemplify standards of achievement in IA, as well as providing some guidance on the general management and teaching of the course.

In computer science, every student has to complete a program dossier that will be internally assessed by the teacher and externally moderated by the IBO. Teachers judge their students' performance by using level descriptors against assessment criteria related to the objectives (see the *Computer Science* guide, pages 55–63). The program dossier is an individual piece of well-documented work, completed during the course, involving a problem that can be solved using computer systems. The emphasis is on the use of a logical approach and analytical thinking, from definition to decomposition of the problem through to its solution by constructing algorithms in Java. This can be carried out in a procedure-oriented or an object-oriented environment.

Section B, "Approaches to the syllabus", is intended to provide further support and guidance by amplifying areas in the *Computer Science* guide, including further example algorithms in Java, and examples of different teaching orders.

This TSM has been written for teachers by examiners and teachers and is intended to provide support and inspiration in a number of ways. It includes:

- sample algorithms in Java
- extracts from past examination papers
- clarifications of the assessment criteria.

The format of this TSM is such that it can be easily amended with replacement pages, for example, further student work, or be supplemented with new materials such as ideas for program dossier problems, lesson plans and teaching suggestions.

The IBO welcomes comments from teachers on this TSM and, also, any ideas for the subject that may be of value to other teachers in IB schools. Comments should be addressed to the subject area manager responsible for computer science, International Baccalaureate Curriculum and Assessment Centre (IBCA), Peterson House, Malthouse Avenue, Cardiff Gate, Cardiff, Wales GB CF23 8GL.

Sample algorithms in Java

Introduction

This section of the support material contains sample algorithms for the International Baccalaureate Diploma Programme (DP) computer science syllabus.

The programs below demonstrate many algorithms contained in the syllabus, as well as some algorithms appropriate for examinations. These examples are **not** intended to show how students should write their program dossiers. Rather, they demonstrate how algorithms are likely to be presented in examinations.

In examinations, algorithms will most likely appear as rather short, single methods (or a few methods)—not as entire programs. Some of the programs below consist of many methods (for example, the RPN calculator). Such algorithms are unlikely to appear in their entirety in examinations.

Notice there is little or no error handling in the code. It illustrates functional algorithms, not good programming techniques. The code should be clear and easily readable, and follow the standards outlined in the JETS section of the syllabus (see the *Computer Science* guide, "Appendix 2", pages 92–116).

The algorithms have been reproduced from the previous (PURE) version published in the *Computer Science* TSM (February 1999). The Java code has been compiled and run successfully under Sun's Java SDK versions 1.3 and 1.4.

Algorithms required by the syllabus

The list of algorithms, which is not exhaustive, represents most of those prescribed within the syllabus, where the tracing or construction of algorithms to perform certain tasks is required. Each algorithm relates to one or more assessment statements (A.S.) in the *Computer Science* guide, and these are also included for reference in this support material to save teachers' time collating algorithms from several sources and rewriting them in Java.

In some cases more than one version of a routine is given, for example bubble sort, and these are labelled version A or version B.

Students are not required to recall the specific implementations given in this support material. Rather, these algorithms are intended to serve as illustrations of specific means of performing a desired task and also to clarify the title, or description, of an algorithm that may have different interpretations in textbooks.

Students may have to trace or construct similar algorithms within examinations, such as being asked to construct a bubble sort for the data supplied in the question.

Function of algorithm	A. S.
Selection sort (version A)	2.1.8
Selection sort (version B)	2.1.8
Bubble sort (version A)	2.1.8
Bubble sort (version B)	2.1.8
Quicksort [HL]	5.2.2
Sequential search	2.1.8
Binary search (version A—iterative)	2.1.8
Binary search (version B—recursive) [HL]	5.5.4

These are the IBIO methods, as given in the *Computer Science* guide. For examination purposes their presence can be assumed.

```
**********************************************************************/

import java.io.*;

public class Algorithms
{  public static void main(String[] args)
   {  new Algorithms();
      System.exit(0);
   }

   public Algorithms()
   {
      char choice = ' ';
      do
      {  output("Please read comments in source code before running.");
         output("----- Standard Sorting and Searching -----");
         output("1 - Selection Sort A");
         output("2 - Selection Sort B");
         output("3 - Bubble Sort A");
         output("4 - Bubble Sort B");
         output("5 - QuickSort");
         output("6 - Sequential Search");
         output("7 - Binary Search Iterative");
         output("8 - Binary Search Recursive");
         output("----- Other Algorithms -------------------");
         output("A - Runtime Errors");
         output("B - Encrypt");
         output("C - Decimal to Binary");
         output("D - Insertion Sort");
         output("E - Binary Search");
         output("F - Round Units");
         output("G - Sort and Remove Duplicates");
         output("H - Valid Pin");
         output("I - Mystery");
         output("J - RPN Calculator");
         output("K - Truth Table");
         output("L - Fractions");
         output("M - Binary Tree");
         output("N - Random Access File");
```

```
         output("O - Indexed File - use Random Access File first to make
data file");
         output("P - Hashing");
         output("Q - Dinner - class hierarchy");
         output("X - Exit");
         choice = inputChar("Choice:");
         if (choice >'Z') { choice = (char)(choice - 32);}    // change to
UPPER CASE
         if (choice >='1' && choice <='5')
         {  int[] nums = {5,3,9,7,1,6,4,8};
            System.out.print("Before sorting:");
            for(int p=0; p < 8; p=p+1){ System.out.print(nums[p]+" ");}
            System.out.println();
            if (choice =='1') {selectionSortA(nums,8);}
            else if (choice =='2') {selectionSortB(nums,8);}
            else if (choice =='3') {bubbleSortA(nums,8);}
            else if (choice =='4') {bubbleSortB(nums,8);}
            else if (choice =='5') {quickSort(nums,0,7);}
            System.out.print("After sorting: ");
            for(int p=0; p < 8; p=p+1){ System.out.print(nums[p]+" ");}
            System.out.println();
         }
         else if (choice >='6' && choice <='8')
         {  int[] nums = {10,20,30,40,50,60,70,80};
            output("The array contains 10,20,30,...,70,80");
            int target = inputInt("Type a number to search for:");
            int pos = -1;
            if (choice =='6') { pos = sequentialSearch(target,nums,8);}
            else if (choice =='7') { pos = binarySearchA(target,nums,8);}
            else if (choice =='8') { pos = binarySearchB(target,nums,0,7);
}
            if (pos >= 0)
            {  output("Found in position " + pos); }
            else
            {  output("Not found"); }
         }
         else if (choice =='A') { runTimeErrors(); }
         else if (choice =='B') { encrypt(); }
         else if (choice =='C') { tryDecToBin(); }
         else if (choice =='D') { tryInsertionSort(); }
         else if (choice =='E') { tryBinarySearch(); }
         else if (choice =='F') { tryRoundUnits(); }
         else if (choice =='G') { trySortAndRemoveDuplicates(); }
         else if (choice =='H') { tryValidPin(); }
         else if (choice =='I') { mystery(); }
         else if (choice =='J') { rpnCalc(); }
         else if (choice =='K') { truthTable(); }
         else if (choice =='L') { fractions(); }
         else if (choice =='M') { binaryTree(); }
         else if (choice =='N') { employees(); }
         else if (choice =='O') { tryIndex(); }
         else if (choice =='P') { tryHashing(); }
         else if (choice =='Q') { new Dinner(); }
         input("-- press [Enter] to continue --");
      } while (choice != 'X');

   }

//============================================================//
```

IBIO standard input and output

```
//=============================================================
//  Below are the IBIO simple input and output methods.
//  These are assumed to be copied into the source code for all
//  algorithms. A note at the end of each algorithm reminds
//  students of this fact. Students are required to
//  understand the USE of these methods, not memorize their code.
//=============================================================
//=============================================================

   static void output(String info)
   { System.out.println(info);   }

   static void output(char info)
   { System.out.println(info);   }

   static void output(byte info)
   { System.out.println(info);   }

   static void output(int info)
   { System.out.println(info);   }

   static void output(long info)
   { System.out.println(info);   }

   static void output(double info)
   { System.out.println(info);   }

   static void output(boolean info)
   { System.out.println(info);   }

   static String input(String prompt)
   {  String inputLine = "";
      System.out.print(prompt);
      try
      {inputLine = (new java.io.BufferedReader(
               new java.io.InputStreamReader(System.in))).readLine();
      }
      catch (Exception e)
      { String err = e.toString();
      System.out.println(err);
      inputLine = "";
      }
      return inputLine;
   }

   static String inputString(String prompt)
   { return input(prompt);   }

   static String input()
   { return input("");       }

   static int inputInt()
   {  return inputInt(""); }

   static double inputDouble()
   { return inputDouble(""); }

   static char inputChar(String prompt)
   {  char result=(char)0;
      try{result=input(prompt).charAt(0);}
```

```
      catch (Exception e){result = (char)0;}
      return result;
   }

   static byte inputByte(String prompt)
   {  byte result=0;
      try{result=Byte.valueOf(input(prompt).trim()).byteValue();}
      catch (Exception e){result = 0;}
      return result;
   }

   static int inputInt(String prompt)
   {  int result=0;
      try{result=Integer.valueOf(
      input(prompt).trim()).intValue();}
      catch (Exception e){result = 0;}
      return result;
   }

   static long inputLong(String prompt)
   {  long result=0;
      try{result=Long.valueOf(input(prompt).trim()).longValue();}
      catch (Exception e){result = 0;}
      return result;
   }

   static double inputDouble(String prompt)
   {  double result=0;
      try{result=Double.valueOf(
      input(prompt).trim()).doubleValue();}
      catch (Exception e){result = 0;}
      return result;
   }

   static boolean inputBoolean(String prompt)
   {  boolean result=false;
      try{result=Boolean.valueOf(
      input(prompt).trim()).booleanValue();}
      catch (Exception e){result = false;}
      return result;
   }
//========== end IBIO ========================================//

}
```

Selection sort (version A)

```
   /*** ************************/

   public void selectionSortA(int[] nums,int size)
   {
      //  Sorts ARRAYOFINT in descending order by finding
      //  the largest element and swapping it into position 1,
      //  then finding the next largest and swapping into 2, etc.

      int first, current, least, temp;

      for(first = 0; first < size; first = first + 1)
      {
         least = first;
         for(current = first+1; current < size; current = current + 1)
         {  if (nums[current] > nums[least])
            {  least = current; }
```

```
        }
        temp = nums[least];
        nums[least] = nums[first];
        nums[first] = temp;
    }
}
```

Selection sort (version B)

```
public void selectionSortB(int[] nums,int size)
{
    //  Same as version A, but sorts in ascending order

    int first, current, least, temp;

    for(first = 0; first < size; first = first + 1)
    {
        least = first;
        for(current = first+1; current < size; current = current + 1)
        {  if (nums[current] < nums[least])
           {  least = current; }
        }
        temp = nums[least];
        nums[least] = nums[first];
        nums[first] = temp;
    }
}
```

Bubble sort (version A)

```
public void bubbleSortA(int[] nums, int size)
{
    //  Sorts into descending order, by comparing direct neighbours and
    //  swapping them if they are out of order.  This performs NUMVALUES
    //  passes.  LAST is decreasing, because a sorted sub-list builds
    //  up at the bottom of the array.

    int last, current, temp;

    for(last = size-1; last > 0; last = last - 1)
    {  for(current = 0; current < last; current = current + 1)
       {  if (nums[current] < nums[current + 1])
          {
             temp = nums[current];
             nums[current] = nums[current+1];
             nums[current+1] = temp;
          }
       }
    }
}
```

Bubble sort (version B)

```
public void bubbleSortB(int[] nums, int size)
{
    //  An ascending bubble sort, but this does not perform a
    //  specific number of passes.  Instead, it stops if an entire pass
    //  completes WITHOUT any swaps occurring.

    int current, temp;
    boolean done;

    do
    {
```

```
        done = true ;
        for(current = 0; current < size-1 ; current = current + 1)
        {  if (nums[current] > nums[current + 1])
           {
              temp = nums[current];
              nums[current] = nums[current+1];
              nums[current+1] = temp;
              done = false;
           }
        }
     } while (!done);
  }
```

Quicksort[HL]

```
  public void quickSort(int[] nums, int start, int finish)
  {
     // A recursive procedure to sort an array into ascending order
     // using the quick-sort algorithm.

     int mid, left, right, temp;

     left = start;
     right = finish;
     mid = nums[(start + finish)/2];  // pivot element chosen from
                                      // the middle of the list
     while (right > left)
     {  while (nums[left] < mid)
        {  left = left + 1;  }
        while (mid < nums[right])
        {  right = right - 1;  }
        if (left <= right)
        {
           temp = nums[left];
           nums[left] = nums[right];
           nums[right] = temp;
           left = left + 1;
           right = right - 1;
        }
     }
     if (start < right)
     {  quickSort(nums, start, right);  }
     if (left < finish)
     {  quickSort(nums, left, finish); }
  }
```

Sequential search

```
  public int sequentialSearch(int target, int[] nums, int size)
  {
     boolean found = false;
     int place = 0;

     while (place < size && !found)
     {
        if (target == nums[place])
        {  found = true; }
        else
        {  place = place + 1; }
     }
```

```
      if (found)
      {  return place; }
      else
      {  return -1;  }
   }
```

Binary search (version A—iterative)

```
   public int binarySearchA(int target, int[] nums, int size)
   {
      //  An iterative binary search.  Size = size of num array.
      //  If found, returns position.  Else returns -1.

      int middle, low, high;
      boolean found = false;

      low = 0;
      high = size-1;
      middle = -1;

      while (high >= low && !found)
      {  middle = (low + high) / 2;
         if (target < nums[middle])
         {  high = middle - 1; }
         else if (target > nums[middle])
         {  low = middle + 1;  }
         else
         {  found = true;  }
      }

      if (found)
      {  return middle; }
      else
      {  return -1;  }
   }
```

Binary search (version B—recursive)[HL]

```
   public int binarySearchB(int target, int[] nums, int low, int high)
   {
      //  A recursive binary search.  Start with low = 0, high = nums.
length-1
      //  If found, returns position.  Else returns -1.

      int middle = (low+high)/2;

      if (low>high)
      {  return -1; }
      else if (target == nums[middle])
      {  return middle; }
      else if (target < nums[middle])
      {  return binarySearchB(target, nums, low, middle - 1); }
      else
      {  return binarySearchB(target, nums, middle+1, high); }
   }
```

Novel algorithms

The IB computer science course expects students to be able to trace or construct novel algorithms that carry out specific tasks, under examination conditions. These particular algorithms will not merely be adapted from those already contained in assessment statements but will be designed to test the ability to transfer skills in order to solve problems.

The algorithms in this part of the support material are intended to illustrate the level of difficulty students can expect when asked to demonstrate these problem-solving skills.

The algorithms are not categorized in any particular way except those that are relevant to HL students only, which are labelled HL.

1 Run-time errors caused by bad input

```
/**************************************************

  Some inputs will cause ERRORS in this algorithm.
  Determine what inputs might cause errors,
  and correct the algorithm to HANDLE these bad inputs properly,
  without run-time exceptions.
**************************************************************************/

  public void runTimeErrors()
  {
     String name = input("What is your name?");
     String abbrev = name.substring(0,3);
     output( "An abbreviation for your name is " + abbrev );

     int num = inputInt("How many children do you have?");
```

```
      int total = 0;
      for (int c=0; c < num; c = c+1)
      {
         int weight = inputInt("Type the weight of child #" + c + ":");
         total = total + weight;
      }
      double average = total / num;
      output("The average weight of your children is " + average);

      String born = input("What month were you born?");
      String[] months = {"Jan","Feb","Mar","Apr","May","Jun",
                         "Jul","Aug","Sep","Oct","Nov","Dec"};
      int m = 0;
      while (!months[m].equals(born))
      { m = m+1; }
      output("That month is # " + m);
   }
/*********************************************************************
   Possible answers for error situations:
    name : any input shorter than 3 letters causes an error in substring
    average : If num is zero, the calculation of average causes
                a division by zero error.
              The calculation of average is usually incorrect, as the
                division operator is polymorphic, and performs an integer
                division for the two integer operands, throwing away
                any decimals in the answer.  But this is a logic error,
                not a run-time error.
    born : The search loop only finds a month if the user types 3 letters.
            If they type the whole name, or misspell the abbreviation,
            the search loop will run past 11 and generate an error.
            This should not be confused with the logic error which
            is caused by the zero-based indexing of the array -
            this algorithm calls "Jan" month # 0.

   Appropriate corrections involve using an if command to prevent
     execution of dangerous code.  For example:

       if (num>0)
       { double average = total / num;
         output("The average weight of your children is " + average);
       }

   It is also appropriate to use good USER INSTRUCTIONS (prompts)
      so the user knows what is expected.  This is especially useful
      when inputting the name of a month.
   Another possibility is error-correction - e.g. using substring
      to extract the first 3 letters of the month name when
      the user types the whole word.
 *********************************************************************
/
```

2 Encrypting a string

```
 ************************************************************
   Show what ENCRYPT does to each of the following strings:
           "test"           "Bobo"
 ************************************************************************
/
   public void encrypt()
   {
      String source,cipher;
      char oldChar, newChar;
```

```
      output("Type in a message, and it will be encrypted.");
      source = input("");
      cipher  =  "";
      for (int c=0; c<source.length(); c=c+1 )
      {
         oldChar = source.charAt(c);
         newChar = ' ';
         if ( (oldChar >= 'a') && (oldChar <= 'z') )
         {
            int ascii = (int)oldChar;
            int pos = ascii - (int)'a';
            int code = ((int)'z') - pos;
            newChar = (char)code;
         }
         else
         {
            newChar = oldChar;
         }
         cipher = newChar + cipher;
      }
      output(cipher);
   }
/**************************************************************************
   Notice that only small letters are encrypted -
    capital letters are not changed.
 *************************************************************************/
```

3 Decimal to binary conversion

```
************************************************

   Convert a decimal integer to binary (returned as a string).
   For negative numbers, returns "negative".
 *************************************************************************
/

   public void tryDecToBin()
   {
      output( "Binary = " + decToBin( inputInt("Type an integer:") ) );
   }

   public String decToBin(int num)
   {  String answer = "";
      if (num < 0)
      { answer = "negative"; }
      else if (num == 0)
      { answer = "0"; }
      else
      {  answer = "";
         while (num > 0)
         {  if (num % 2 == 0)
            {  answer = "0" + answer; }
            else
            {  answer = "1" + answer; }
            num = num / 2;
         }
      }
      return answer;
   }
```

4 Insertion sort

```
**************************************************

  Takes each element, and moves it upward through the list until
  reaching the correct position, stopping when it "bumps" against
  a smaller value. The resulting list is in ascending order.
**********************************************************************/


  public void tryInsertionSort()
  {
     int[] nums = {15,7,20,13,25,18};
     for (int c=0; c < nums.length; c = c+1)
     {  output(c + " : " + nums[c]);  }
     output("Sorting");
     insertionSort(nums);
     for (int c=0; c < nums.length; c = c+1)
     {  output(c + " : " + nums[c]);  }
  }

  public void insertionSort( int[] nums)
  {
     for (int newest = 1; newest < nums.length; newest++)
     {  int newValue = nums[newest];
        int current = newest;
        while ( (current > 0) && (nums[current - 1] > newValue) )
        {  nums[current] = nums[current - 1];
           current = current - 1;
       }
        nums[current] = newValue;
     }
  }
```

5 Binary search (iterative function)

```
**************************************************

  in a sorted array (iterative, non-recursive version)
  Returns the position of the desired string.
  Returns -1 if the string is not found.
**********************************************************************/

  public void tryBinarySearch()
  {
     String[] names = {"Alpha","Beta","Epsilon","Delta",
                       "Gamma","Lambda","Mu","Omega"};
     output("The list is:");
     for (int c=0; c < names.length; c = c+1)
     {  output( c + " : " + names[c]); }
     int lambda = binarySearch(names,"Lambda");
     output("Lambda found in position " + lambda);
     int camel = binarySearch(names,"Camel");
     output("Camel found in position " + camel);
     int omega = binarySearch(names,"Omega");
     output("Omega found in position " + omega);
  }


  public int binarySearch(String[] list,String find)
  {
     int mid = 0;
     int low = 0;
```

```
      int high = list.length-1;
      boolean found = false;

      while ( !found && (high >= low))
      {  mid = (low + high) / 2;
         if ( find.compareTo(list[mid]) < 0 )
         {  high = mid - 1; }
         else if (find.compareTo(list[mid]) > 0)
         {  low = mid + 1;  }
         else
         {  found = true; }

      }
      if (!found)
      {  return -1; }
      else
      {  return mid; }
   }
```

6 Rounding numbers to SI prefixes

```
 ************************************************************

   Converts a number with standard SI prefix T,G,M,k,c,m,u,n rounded to
   an integer multiple of the unit. For example,
     ROUNDUNITS(1234567890,"M") returns "1235 M"
     ROUNDUNITS(1234567890,"G") returns "1 G"
     ROUNDUNITS(0.02,"m")       returns "20 m"
     ROUNDUNITS(1000,"M")       returns "0 M"
   This conversion uses powers of 10, not powers of 2, e.g. k = 1000
   not k = 1024. For printing convenience, 'u' represents micro (10^-6).
 ************************************************************************
/

   public void tryRoundUnits()
   {
      output( 1.23 + " = " + roundUnits(1.23,'m') );
      output( 1.23 + " = " + roundUnits(1.23,'c') );
      output( 123456789 + " = " + roundUnits(123456789,'M') );
   }

   public String roundUnits(double num, char unit)
   {
      String units = "TGMKcmun";
      double[] values = new double[8];
      int x,p;
      double v = 1000;
      String result = "";

      for(x = 3; x >= 0; x = x-1)
      {
         values[x] = v;
         v = v * 1000;
      }

      values[4] = 0.01;
      v = 0.001;
      for(x = 5; x < 8; x = x+1)
      {  values[x] = v;
         v = v / 1000;
      }

      for(x = 0; x < 8; x = x+1)
```

```
      {
         if (units.charAt(x) == unit)
         {  v = Math.round(num/values[x]);
            result = v + " " + unit;
         }
      }
      return(result);
   }
```

7 Sorting and removing duplicates from an array

```
 ****************************************

   Sort and remove duplicate entries from a linear array.
   Inputs names until 'xxx' is typed. CLEANUP sorts the LIST, and then
   removes any duplicates.  Returns a new copy of the array, but
   shorter if items were removed.
 ************************************************************************
/

   public void trySortAndRemoveDuplicates()
   {
      String[] names = new String[5];
      output("Type 5 names:");
      for (int c = 0; c < 5 ; c = c + 1)
      {  names[c] = input(c + ":");  }
      names = sortAndRemoveDuplicates(names);
      output("After removing duplicates and sorting:");
      for (int c = 0; c < names.length; c = c + 1)
      {  output(c + " : " + names[c]);  }
   }

   public String[] sortAndRemoveDuplicates(String[] list)
   {  if (list.length == 0)
      { return(list); }

      //---- Bubble Sort -------------------------------------------
      boolean swapped = true;
      int     pass = 1;
      do
      {  swapped = false;
         for(int x = 0; x < list.length - pass; x++)
         {
            if (list[x].compareTo(list[x + 1]) > 0)
            {
               String temp = list[x];        // 3-way swap
               list[x] = list[x + 1];
               list[x+ 1] = temp;
               swapped = true;               // swapped, need another pass
            }
         }
         pass = pass + 1;
      } while (swapped);                     // Quits if no swaps occurred

      //---- Count non-duplicates ------------------------
      int unique = 1;
      for (int x=0; x < list.length-1; x = x+1)
      {  if (!list[x].equals(list[x+1]))        // counts if this item is
         { unique = unique + 1;}                // different from next item
      }

      //---- Copy unique items into new results array -----
      String[] results = new String[unique];
```

```
      int p = 0;
      for (int x=0;x < list.length-1; x = x+1)
      {  if ( !list[x].equals(list[x+1]) )
         {  results[p] = list[x];
            p = p+1;
         }
      }
      results[p] = list[list.length-1];         // last item always counts
      return results;

   }
```

8 Check digit validation

```
 ****************************************

   This algorithm validates a decimal integer PIN (Personal
   Identification Number) according to the rule:

   a PIN number is valid if the sum of all the digits has the
   same last digit as the product of all the non-zero digits.

   For example, 64 is not valid, because 6*4 = 24 which ends in a 4,
   but 6+4=10 which ends in 0. However, it is easy to turn 64 into a
   valid PIN number, by adding 1s (ones) until the sum and product
   match. Thus, 116141 is valid, as the sum of the digits is 14, and
   the product is 24. Since zeroes do not change the product or the
   sum, 1016010401 is also valid.
 ************************************************************************
/

   public void tryValidPin()
   {
      int pin = 0;
      do
      {
         pin = inputInt("Pin:");
         output( validPin(pin) );
      } while (!validPin(pin) && pin>0);
   }

   public boolean validPin(int pin)
   {
       int sum = 0;
       int product = 1;

      while (pin>0)
      {
         int digit = pin % 10;
         sum = sum + digit;
         if (digit != 0)
         {
             product = product * digit;
         }
         pin = pin / 10;
      }
      if ( (sum % 10) == (product % 10) )
      { return true; }
      else
      { return false; }
   }
```

9 Tracing a mystery algorithm

```
 ************************************************************

   What is printed by SHOW the first time, and what the second time?
 ************************************************************************
/
   int size;
   int[][] nums = new int[4][4];

   public void mystery()
   {  start();
      output("- Original -");
      show();
      change();
      output("- Changed -");
      show();
   }

   public void start()
   {
      int row,col,count;
      size = 4;
      count = 0;
      for (row = 0; row < size; row = row + 1)
      {  for (col = 0; col < size; col = col + 1)
         {  count = count + 1;
            if (row < col)
            {  nums[row][col] = row; }
            else
            {  nums[row][col] = count;}

         }
      }
   }

   public void show()
   {
      int row,col,count;
      for (row = 0;row < size; row = row + 1)
      {  for (col = 0; col < size; col = col + 1)
         {  System.out.print(nums[row][col] + "\t"); }
         System.out.println("");
      }
   }

   public void change()
   {
     int row,col,temp,half;
     half = size / 2;
     for (col = 0; col < half; col = col + 1)
     {
        for (row = 0; row < size; row = row + 1)
        {
           temp = nums[row][col];
           nums[row][col] = nums[row][size - 1 - col];
           nums[row][size - 1 - col] = temp;
        }
     }
   }
```

10 RPN calculator using a stack [HL]

```
************************************************

  A Reverse Polish Notation calculator that uses a dynamic stack.
  It uses the value method to convert strings into numbers.
  It uses a Dynamic Stack stored in a Linked-List.
*********************************************************************/

  public double value(String s)
  {
     // Changes a string into the equivalent double value
     // It returns 0 if the conversion causes an error.

     double answer = 0;
     try
     {  answer=Double.valueOf(s.trim()).doubleValue();
     }
     catch (Exception e) {};
     return answer;
  }


  class ListNode          // This inner class is used as a data structure
  {
     double data;
     ListNode next;
  }

  int error;

  ListNode stack = null;

  public void rpnCalc()
  {
     String term;
     double number = 0, answer = 0;
     String[] messages = {"No error",
                          "Division by zero",
                          "Stack underflow",
                          "Memory overflow"};
     error = -1;
     while (error < 0)
     {
        term = input("Next term:");
        if (term.equals("=") )
        {  error = 0;
           answer = pop();
        }
        else if (term.equals("+"))
        {  push(pop() + pop()); }
        else if (term.equals("-"))
        {  push(-1*pop() + pop()); }
        else if (term.equals("*"))
        {  push(pop() * pop()); }
        else if (term.equals("/"))
        {  number = pop();
           if (number == 0)
           {  error = 1; }
           else
           { push(pop()/number); }
        }
```

```
         else
         {  push(value(term));  }
      }
      if (error == 0)
      {  output("The result is " + answer); }
      else
      {  output("ERROR : " + messages[error]); }
   }

   public double pop()
   {
      double number;
      if (stack == null)
      {  error = 2;
         number = 0;
      }
      else
      {  number = stack.data;
         stack = stack.next;
      }
      return number;
   }

   public void push(double number)
   {
      ListNode temp = new ListNode();
      if (temp == null)
      {  error = 3; }
      else
      {
         temp.data = number;
         temp.next = stack;
         stack = temp;
      }
   }
```

11 Formatted truth table [HL]

```
 ********************************************************

   Generate a truth table for a Boolean expression.
   Uses escaped-TAB characters to line up columns.
 ************************************************************************
/

   public void truthTable()
   {
      boolean A,B,C,truth;
      output(" A\tB\tC\t(A and not B) or not(B and C)");
      A = false;
      do
      {
         B = false;
         do
         {
            C = false;
            do
            {
               truth = (A && !B) || !(B && C) ;
               output(A + "\t" + B + "\t" + C + "\t" + truth);
               C = !C;
            } while (C);
            B = !B;
```

```
         } while (B);
         A = !A;
      }  while (A);
   }

/*************** Sample Output ***************************************

    A       B       C      (A and not B) or not(B and C)
   false   false   false   true
   false   false   true    true
   false   true    false   true
   false   true    true    false
   true    false   false   true
   true    false   true    true
   true    true    false   true
   true    true    true    false

 ********************************************************************
```

12 Fractions

```
****/


 ************************************************

   An example of using an "inner class" to store several
   data fields - similar to a "record" structure in a
   traditional language.  Notice that methods can return a
   result as a new Fraction, or by changing the values in
   the input parameter object.  In Java, all parameters
   are passed by value.  If an object is passed as a parameter,
   its value (address) cannot be changed, but the contents
   of the object it points to CAN be changed.

   A better version of this algorithm would prevent the user
   from entering 0 as the bottom of a fraction.
 ************************************************************/

   class Fraction
   {
      int top;
      int bottom;

      public Fraction(int t, int b)
      {
         top = t;
         bottom = b;
      }
   }

   public void fractions()
   {
      Fraction fa = inputFraction("First fraction:");
      Fraction fb = inputFraction("Second fraction:");
      Fraction fc = addFractions(fa,fb);
      reduce(fc);
      output("Sum = " + fc.top + "/" + fc.bottom);
   }

   public Fraction inputFraction(String prompt)
   {
      output(prompt);
```



```
      return new Fraction(inputInt(" Top = "), inputInt(" Bottom = ") );
   }

   public Fraction addFractions(Fraction fa, Fraction fb)
   {
      int top = fa.top * fb.bottom + fa.bottom * fb.top;
      int bottom = fa.bottom * fb.bottom;
      return new Fraction( top, bottom );
   }

   public void reduce(Fraction frac)
   {
      if (frac.top != 0)
      {  int f = frac.top;
         while ( (frac.top % f != 0) || (frac.bottom % f != 0) )
         {
            f = f - 1;
         }
         frac.top = frac.top / f ;
         frac.bottom = frac.bottom / f ;
      }
   }
```

13 Binary tree [HL]

```
 ************************************************************

   Reads strings from the keyboard, in any order. Each word is
   inserted into the binary search tree in the proper position.
   Then an in-order traversal prints the words in alphabetical order.
   This concept could be used to sort a text file, if the input came
   from a text file and the output went back into the same file.
 ************************************************************************
/

   class TreeNode
   {
      String data;
      TreeNode leftChild;
      TreeNode rightChild;

      public TreeNode(String info)
      {
         data = info;
         leftChild = null;
         rightChild = null;
      }
   }

   TreeNode root = null;

   public void binaryTree()
   {
      makeTree();
      showTree(root);
   }

   public void makeTree()
   {
      root = null;
      String word = "";
      do
      {
```

```
      word = input("Type a word (xxx to quit):");
      if (!word.equals("xxx"))
      {  addTreeNode(word);  }
   } while(!word.equals("xxx"));
}

public void addTreeNode(String word)
{
   boolean done = false;
   if (root == null)
   {
      root = new TreeNode(word);
      done = true;
   }

   TreeNode temp = root;
   while (!done)
   {
      if (temp.data.equals(word))
      {  done = true;  }             // word found, so don't add
      else if (word.compareTo(temp.data)<0)
      {
         if (temp.leftChild == null)
         {  temp.leftChild = new TreeNode(word);
            done = true;
         }
         else
         {  temp = temp.leftChild;  }
      }
      else
      {
         if (temp.rightChild == null)
         {  temp.rightChild = new TreeNode(word);
            done = true;
         }
         else
         {  temp = temp.rightChild;  }
      }
   }
   return;
}

public void showTree(TreeNode here)
{
   if (here == null)
   {  return;  }
   else
   {
      showTree(here.leftChild);
      output(here.data);
      showTree(here.rightChild);
   }
}
```

14 Random access file [HL]

```
*****************************************

  Creates a RandomAccessFile with names and salaries.
  Records are added to the file in order as they are input.
  Then a bubble sort exchanges records to put the salaries in order.
  At the end, it prints the the file.
*****************************************************************/

  public void employees()
  {
     inputEmployees();
     sortEmployees();
     outputEmployees();
  }

  public void clearFile()
  {
     try
     {
        RandomAccessFile file = new RandomAccessFile("employees.dat","rw");
        file.setLength(0);
        file.close();
     }
     catch (IOException e) { output(e.toString()); }
  }

  public void writeName(long record,String name)
  {  try
     {  RandomAccessFile file = new RandomAccessFile("employees.dat","rw");
        if (name.length() > 40)
        { name = name.substring(0,40);}
        file.seek(50*record);
        file.writeUTF(name);
        file.close();
     }
     catch(IOException e) { output(e.toString()); }
  }

  public void writeSalary(long record,double salary)
  {
     try
     {
        RandomAccessFile file = new RandomAccessFile("employees.dat","rw");
        file.seek(50*record+42);
        file.writeDouble(salary);
        file.close();
     }
     catch (IOException e) { output(e.toString()); }
  }

  public long countRecords()
  {
     try
     {
        RandomAccessFile file = new RandomAccessFile("employees.dat","r");
        long records = file.length() / 50;
        file.close();
        return records;
     }
     catch (IOException e) { return -1; }
```

```
}

public String readName(long record)
{
   String result=null;
   try
   {  RandomAccessFile file = new RandomAccessFile("employees.dat","r");
      if (record < countRecords())
      {
         file.seek(50*record);
         result = file.readUTF();
      }
      else
      {
         result = null;
      }
      file.close();
   }
   catch(IOException e) { output(e.toString()); }
   return result;
}

public double readSalary(long record)
{
   double result=-1;
   try
   {  RandomAccessFile file = new RandomAccessFile("employees.dat","r");
      if (record < countRecords())
      {
         file.seek(50*record+42);
         result = file.readDouble();
      }
      else
      {
         result = -1;
      }
      file.close();
   }
   catch(IOException e) { output(e.toString()); }
   return result;
}

public void inputEmployees()
{
   clearFile();
   long record = 0;
   String name = "";
   double salary = 0;
   while (salary >= 0 )
   {
      name = input("Name:");
      salary = inputDouble("Salary (-1 to quit):");
      if (salary >= 0)
      {
         writeName(record,name);
         writeSalary(record,salary);
         record = record + 1;
      }
   }
}

public void sortEmployees()
```

```
{ try
  {
   long recordCount = countRecords();
   for (long pass = 0; pass < recordCount-1; pass = pass + 1)
   {
      for (long c = 0; c < recordCount-1; c = c + 1)
      {
         String nameA = readName(c);
         double salaryA = readSalary(c);
         String nameB = readName(c+1);
         double salaryB = readSalary(c+1);
         if (salaryB > salaryA)
         {
            writeName(c,nameB);
            writeSalary(c,salaryB);
            writeName(c+1,nameA);
            writeSalary(c+1,salaryA);
         }
      }
   }
  }
   catch (Exception e){output(e.toString());return;}
}

public void outputEmployees()
{
   for (long c = 0; c < countRecords(); c = c+1)
   {
      String name = readName(c);
      double salary = readSalary(c);
      output(name + "\t" + salary);
   }
}
```

15 Fully indexed file [HL]

```
*****************************************

  The class IndexedFile creates a FULL INDEX for the NAME field
  in the RandomAccessFile employees.dat.  There is an entry in the
  KEY array for each record in the file.  The index is created when
  the class is instantiated.  This is an oversimplified example,
  with a fixed-size array.  It also should permit adding new
  records to the file, but this is not included.  It also assumes
  the file is named "employees.dat" and records have 2 fields:
  a 40-character UTF name and a double salary.
  The sorting algorithm is inefficient.

  The getSalary method retrieves the salary by
  searching for an employee's name in the key[] array,
  following the matching pos[] pointer, and retrieving
  the salary from the file.
*****************************************************************/

  class IndexedFile
  {
     String[] key = new String[1000];
     long[] pos = new long[1000];
     int size = 0;

     public IndexedFile()
     {
```

```
      // read names into key[] array
      try
      {
        RandomAccessFile file = new RandomAccessFile("employees.dat","r");
         long records = file.length() / 50;
         for (long c=0; c < records; c = c+1)
         {
            file.seek(c*50);
            key[size] = file.readUTF();
            file.seek(c*50+42);
            pos[size] = c;
            size = size + 1;
         }
         file.close();
      }
      catch (IOException e){ output(e.toString()); }

      // sort key[] array, moving pos[] entries to
      //  stay matched with key[] names
      for (int pass = 0; pass < size; pass = pass + 1)
      {
         for (int c = 0; c < size-1; c = c + 1)
         {
            if (key[c].compareTo(key[c+1])>0)
            {
               String tempKey = key[c];
               long tempPos = pos[c];
               key[c] = key[c+1];
               pos[c] = pos[c+1];
               key[c+1] = tempKey;
               pos[c+1] = tempPos;
            }
         }
      }
   }

   public double getSalary(String name)
   {
      double salary = -1;
      int lo = 0;
      int hi = size-1;
      int found = -1;
      while ( (hi >= lo) && (found < 0) )
      {
         int mid = (lo + hi) / 2;
         if ( key[mid].equals(name) )
         {  found = mid; }
         else if ( name.compareTo(key[mid]) < 0)
         {  hi = mid - 1; }
         else
         {  lo = mid + 1; }
      }
      if (found >= 0)
      {  try
         {
            RandomAccessFile file = new RandomAccessFile("employees.dat","r");
            file.seek(pos[found]*50+42);
            salary = file.readDouble();
            file.close();
         }
         catch(IOException e) { output(e.toString()); }
      }
```

```
        return salary;
     }
  }

  public void tryIndex()
  {
     String name;
     IndexedFile allNames = new IndexedFile();
     output("The file contains these names:");
     for(int x=0; x < allNames.size; x=x+1)
     {  output(allNames.key[x]); }
     do
     {
        name = input("Type a name (xxx to quit):");
        if (!name.equals("xxx"))
        {
           IndexedFile iFile = new IndexedFile();
           double salary = iFile.getSalary(name);
           if (salary>=0)
           {  output( "Salary = " + salary ); }
           else
           {  output( "Not found" ); }
        }
     } while (!name.equals("xxx"));
  }
```

16 Hashing methods (data file) HL

```
*************************************************

   Creates random "words" by putting together random letters.
   Stores the words in a Hash-Table.  The Hash Code is
   calculated from the ASCII codes of the characters in the
   word.  If the corresponding position is full, the putHash
   method searches sequentially for the next free position
   and stores the word there.
*************************************************************/

  public void tryHashing()
  {
     String[] data = new String[13];      // primes are LUCKY
     for (int c=0; c < 13; c = c+1)
     {  data[c] = ""; }
     for (int c=0; c < 10; c = c+1)
     {
        String word = randomConsonant() + randomVowel() +
              randomConsonant() + randomConsonant() + randomVowel();
        hashPut(data,word);
     }
     for (int c=0; c < 13; c = c+1)
     {  output(c + " : " + data[c]); }
  }

  public String randomConsonant()
  {
     String consonants = "BCDFGHJKLMNPQRSTVWXZ";
     int count = consonants.length();
     int random = (int)(Math.random()*count);
     return consonants.substring(random,random+1);
  }
```

```
public String randomVowel()
{
   String vowels = "AEIOUY";
   int count = vowels.length();
   int random = (int)(Math.random()*count);
   return vowels.substring(random,random+1);
}

public void hashPut(String[] data,String word)
{
   int pos = hashCode(word);
   int full = 0;
   while ( (!data[pos].equals("")) && (full < 13) )
   {
      full = full + 1;  // searching for empty space
      pos = pos+1;
      if (pos >= 13)
      {  pos = 0; }     // wrap-around at end of array
   }
   if (full < 13)
   {  data[pos] = word; }
   else
   {  output("Data array is FULL"); }
}

public int hashCode(String word)
{
   int code = word.length();
   if (word.length()>0)
   {  code = code + 7*word.charAt(0); }
   if (word.length() > 1)
   {  code = code + 5*word.charAt(1); }
   if (word.length() > 2)
   {  code = code + 3*word.charAt(2); }
   return code % 13;
}
```

17 Dinner tables—class decomposition HL

```
*******************************

Dinner is used to plan the seating of guests at tables for
a formal dinner party.  It uses several classes.
Some error trapping has been done, but some run-time errors
can still occur.

- Class Decomposition -

In Object-Oriented programming (e.g. Java), we try to use
CLASS decomposition to structure the solution.  This decomposition
should be similar in both the design and implementation sections.
In this problem, the identifiable objects are:

   Guests
   Tables
   Dining-room (a list of tables)
   Dinner event.

They are hierarchically related like this:

   Dinner
```

```
         TableList
            Table, Table, Table
         GuestList.

 ************************************************************************
/

   class Dinner
   {
      GuestList guests ;
      TableList tables ;

      public Dinner()
      {
         inputGuests();
         output("-------");
         inputTables();
         output("-------");

         String done;
         do
         {
            assignTables();
            output("-------");
            output("- Guests -\n" + guests.toString());
            output("- Tables -\n" + tables.toString());
            done = input("Are you finished (Y/N)?");
         } while (done.equals("N") || done.equals("n"));
      }

      public void inputGuests()
      {
         int maxGuests = inputInt("How many guests maximum?");
         guests = new GuestList(maxGuests);

         String newName = "";
         output("Type the guest names (you can still add more later)");
         do
         {
            newName = input("Name (or QUIT):");
            if (!newName.equals("QUIT"))
            {  guests.add(newName); }
         }  while (!newName.equals("QUIT"));
      }

      public void inputTables()
      {  int maxTables = inputInt("How many tables?");
         int maxSeats  = inputInt("How many seats at each table?");

         tables = new TableList(maxTables, maxSeats);
      }

      public void assignTables()
      {
         output("Assigning Tables");
         String name;
         int table;
         do
         {  name = input("Name of guest (or QUIT):");
            if ( (guests.find(name) < 0) && (!name.equals("QUIT")))
```

```
            {  String addYN = input("Name not found - add it (Y/N)");
               if (addYN.equals("Y"))
               { guests.add(name);}
            }
            if (guests.find(name)>=0)
            {  int max = tables.tables.length - 1;
               table = inputInt("Number of table (0-" + max +"):");
               if ( table >=0 )
               { String result = tables.reserve(name,table);
                 if (result.equals(""))
                 {  guests.assign(name,table);}
                 else
                 {  output( result ); }
               }
            }
         }  while (!name.equals("QUIT"));
      }
   }

   class TableList
   {
      Table[] tables ;    // List of tables

      public TableList(int maxTables, int maxSeats)
      {
         tables = new Table[maxTables];
         for (int c = 0; c < maxTables; c = c+1)
         {  tables[c] = new Table(maxSeats); }
      }

      public String reserve(String name,int tableNum)
      {
         if (tables[tableNum].available() > 0)
         {  tables[tableNum].assign(name);
            return "";
         }
         else
         {  return "Table FULL"; }
      }

      public String toString()
      {
         String result = "";
         for (int t = 0; t < tables.length; t = t+1)
         {  result = result + t + ": " + tables[t].toString() + "\n"; }
         return result;
      }
   }

   class Table
   {
      String[] names;   // names of guests assigned to this table
      int count ;       // number of guests assigned

      public Table(int maxSeats)
      {
         names = new String[maxSeats];
         for (int c = 0; c < maxSeats; c = c+1)
         {  names[c] = "---";  }
         count = 0;
      }
```

```
   public boolean assign(String name)
   {
      if (count < names.length)
      {
         names[count] = name;
         count = count + 1;
         return true;
      }
      else
      {
         return false;
      }
   }

   public int available()
   {
      return names.length - count;
   }

   public String toString()
   {
      String result = "";
      for (int c = 0; c < names.length ; c = c + 1)
      {  result = result + names[c] + "  "; }
      return result;
   }
}

class GuestList
{
   String[] names;      // names of guests
   int[] seating;       // seats assigned
   int count;

   public GuestList(int maxGuests)    // constructor
   {
      names = new String[maxGuests];
      seating = new int[maxGuests];

      count = 0;
   }

   public int find(String name)       // search for a name in names[]
   {  int found = -1;
      for (int c = 0; c < count; c = c+1)
      {  if(names[c].equals(name))
         {  found = c;  }
      }
      return found;                   // return -1 if not found
   }

   public void assign(String name,int tableNum)
   {                                  // assign name to tableNum
      int pos = find(name);
      if (pos >=0 )
      {  seating[pos] = tableNum;  }  // put tableNum into seating[]
   }

   public void add(String name)       // add a new name into names[]
   {
      if (count < names.length)
      {
```

```
            names[count] = name;
            seating[count] = -1;          // seat is still unassigned
            count = count + 1;
         }
         else
         {  System.out.println("Guest list is FULL"); }
      }

      public String toString()           // create a string containing
      {  String result = "";             //  all names and seats
         for (int c = 0; c < count; c = c+1)
         {  result = result + names[c] + " at " + seating[c] + "\n"; }
         return result;
      }

   }
```

Algorithms from IB examinations

Previous examination paper questions, even those that apply to an earlier syllabus, can be a valuable resource because they can be used:

- to illustrate algorithms that relate directly to the syllabus details
- to strengthen problem-solving skills
- as a source of data and information for school-based tests.

This section consists of examples of questions that required students to trace and/or construct algorithms and their solutions. Some are complete questions, and others are parts of questions; they both refer to algorithm tracing or construction. In some cases solutions have been provided because they include an algorithm as part of the answer.

All questions and solutions have been rewritten in Java.

The questions are categorized according to year, level, paper, question number, the action on the algorithm required, and the content area of the question.

No.	Year	Level & paper	Qu.	Action on algorithm	Content of question
1	M96	SL P1	1	Trace	Summing and averaging
2	M96	SL P1	3	Trace	Counting loops
3	M96	SL P2	1	Trace and construct	Determining frequencies
4	M96	SL P2	5	Construct	**If-then-else** statements
5	N96	SL P1	1	Trace	Dividing and truncating to manipulate single-decimal digits
6	N96	SL P1	3	Trace and construct	Nonsense with absolute value
7	N96	SL P2	5	Trace and construct	Calculating typing fees
8	M97	SL P1	1	Trace and analyse	Golf scores in arrays
9	M97	SL P1	3	Trace	Summing loop
10	M97	SL P2	1	Construct	One-dimensional array manipulation (hashing)
11	N97	SL P1	1	Trace	Finding the largest value in an array
12	N97	SL P1	3	Trace and modify	Binary search
13	N97	SL P2	5	Construct	Simulation to count fish
14	M96	HL P1	10	Trace and construct	Highest values in a two-dimensional array
15	M96	HL P2	4	Trace and construct	Fibonacci sequence (recursive)
16	N96	HL P2	1	Trace and construct	Binary search
17	N96	HL P2	7	Trace and modify	Binary tree
18	N97	HL P1	10	Trace and construct	Summing in a two-dimensional array
19	N97	HL P2	1	Trace and construct	Random numbers

1 Summing and averaging (M96 SL P1 Q1)

The following algorithm fragment has been designed to analyse the temperatures at a tourist resort.

```
int count, total, temp, big;
double average;

count = 0;                                    // 1
total = 0;                                    // 2
temp = inputInt("Type a number");             // 3
big = temp;                                   // 4
while (temp != 0)                             // 5
{    total = total + temp;                    // 6
     count = count + 1;                       // 7
     temp = inputInt("Type a number");        // 8
     if (temp > big)                          // 9
     {     big = temp;                        // 10
     }                                        // 11
}                                             // 12
average = total / count;                      // 13
output(average + " " + big);                  // 14
```

Copy and complete the following trace table for the data: 15, 7, 23, 9, 0

Line	COUNT	TOTAL	TEMP	BIG	TEMP # 0	**output**
1	0					
2		0				
3			15			
4				15		
5					true	
6						

2 Counting loops (M96 SL P1 Q3)

(a) Consider the algorithm fragment below.

```
int x,y;

x = 1;
while (x < 6)
{
      y = 1;
      while (y < 5)
      {
            y = y + 1;
      }
      output("The product of X and Y is " + x * y);
      output("The values of X and Y are " + x + " " + y);
      x = x + 1;
}

.....
```

Complete the trace through the fragment. The first two lines are provided and your output should be similar to that below:

The product of X and Y is 5

The values of X and Y are 1 5

.

.

.

(b) Consider the algorithm fragment below.

```
int x,y;

x = 1;
while (x < 6)
{
      y = 1;
      while (y < 6)
      {
            output("The product of x and y is " + x * y);
            y = y + 2;
      }
      output("The values of X and Y are " + x + " " + y);
      x = x + 2;
}
```

Complete the trace through the fragment. The first line is provided and your output should be similar to that below:

The product of X and Y is 1

.

.

.

3 Determining frequencies (M96 SL P2 Q1)

In your school, you must determine the number of students taking various examinations. As part of the input process, a student enters the code of a subject into a computer. For example, 21 could represent computer science. These are stored in an array and the frequency (number of times) with which each code occurs is to be determined.

(a) Consider the following subject code data:

DATA: 21 14 25 23 21 21 25 14 16 16 17 21 25 17 21 23 14 25 17

Copy and complete the tables below to show the unique subject codes and their corresponding frequency. Use the subject code data above.

CODES	FREQS
21	

(b) Consider the following variables:

CODES	Single-dimension array of integer values that will eventually contain the subject codes without duplications. (Initialized for a maximum of 750 values.)
FREQS	Single-dimension array of integer values that will eventually contain the frequency of the subject codes. (Initialized for a maximum of 750 values.)
SIZE	Integer variable that indicates the current number of valid entries in CODES and FREQS.
CODE	Integer variable that contains an input subject code.
FOUND	Boolean variable that indicates if the current value in CODE is already stored in CODES.
POSITION	Integer variable that indicates the position that CODE should be placed in the CODES array.

The algorithm below (FREQUENCIES) uses these variables to compute the subject code frequencies.

```
public frequencies()
{
      int  size, code, position;
      int[]  codes = new int[750];
      int[]  freqs = new int[750];
      boolean  found;

      size = 0;

      code = inputInt();   // the input stream is terminated by -99

      while ((size <= 750) && (code != -99))
      {
            position = search(codes, size, code);

            if (position < 0)
           {
                        . . . .
                        . . . .
                        . . . .
           }
      }

}
```

(i) The subalgorithm SEARCH tests if the current subject code (stored in CODE) is already in array CODES. Explain the values returned for position if:

- CODE is already in array CODES

- CODE is not already in array CODES.

(ii) Construct subalgorithm SEARCH.

(iii) Explain how the contents of FREQS and CODES will be updated by the algorithm FREQUENCIES if:

- CODE is already in array CODES

- CODE is not already in array CODES.

(iv) Complete the algorithm FREQUENCIES.

A possible solution for (b) (ii)

```
int SEARCH(int[] CODES, int SIZE, int code)
{
      int  X;

      boolean found = false;
      x = 0;
      while ((X < SIZE) && (!FOUND) )
      {
            X = X + 1;
            if (CODES[X] == CODE)
            {     FOUND = true;  }
      }
      if (found)
      {return x; }
      else
      { return -1; }
}
```

A possible solution for (b) (iv)

```
public frequencies()
{
      int  size, code, position;
      int[]  codes = new int[750];
      int[]  freqs = new int[750];
      boolean  found;

      size = 0;

      code = inputInt();   // the input stream is terminated by -99

      while ((size <= 750) && (code != -99))
      {
            position = search(codes, size, code);

            if (position < 0)
           {
               size <-- size + 1;
               codes[size] <-- code;
               freqs[size] <-- 1;
            }
            else
            {  freqs[position] = freqs[position] + 1; }

      }
}
```

4 If-then-else statements (M96 SL P2 Q5)

When evaluating the Boolean operators, **and** and **or**, in some circumstances the evaluation of the entire Boolean expression can be determined by the value of the first operand.

For example, the statement "S1 **&&** S2" is false if S1 is false. Similarly, "S1 **||** S2" is true if S1 is true. In both of these cases, the value of S2 is irrelevant.

Rewrite the following algorithm fragments to use this strategy, testing only one operand at a time (for example, **if** S1 **...** , **if !** S1 **...**).

(a) Implement the statement below. No Boolean operators are necessary.

```
if (S1 || S2)
{     ACTION1(); }
else
{     ACTION2(); }
```

(b) Implement the statement below.

```
if (S1 && S2)
{     ACTION1(); }
else
{     ACTION2(); }
```

A possible solution for (a):

```
if (S1)
{     ACTION1(); }
else
{     if (S2)
      {     ACTION1(); }
      else
      {     ACTION2(); }
}
```

A possible solution for (b):

```
if (S1)
{     if (S2)
      {  ACTION1(); }
      else
      {  ACTION2(); }
}
else
{  ACTION2(); }
```

5 Dividing and truncating to manipulate single-decimal digits (N96 SL P1 Q1)

Consider the procedure below.

```
public void alter(int number)
{     int  newValue, digit;

1     newValue = 0;
2     while (number > 0)
3     {  digit = number - (int)(number / 10) * 10;
4        newValue = newValue * 10 + digit;
5        number = (int)(number / 10);
6     }
7     output( newValue );

}
```

Remember that **(int)**(769.84) returns 769.

.....

(a) Copy and complete the following trace table for the call ALTER(123).

Line	NUMBER	NEWVALUE	NUMBER > 0	DIGIT	**output**
	123				
1		0			
2			true		
3				3	

(b) Describe the purpose of the procedure ALTER.

(c) State the output for the call ALTER(-123).

(d) Explain the difference if ALTER used a pass-by-reference parameter rather than a pass-by-value parameter.

6 Nonsense with absolute value (N96 SL P1 Q3)

Consider the algorithm fragment below. Remember that **Math.abs**(- 23.4) returns +23.4 and **Math.abs** (1051.0) returns +1051.0.

```
double number1, number2;

number1 = inputDouble("Type a number:");
if (number1 >= 0.0)
{     if ( number1 < 1000.0 )
      {     number2 = 2 * number1;
            if (number1 <= 500.0)
            {     number1 = number1 / 10.0; }
      }
      else
      {     number2 = 3 * number1; }
}
else
{     number2 = Math.abs(number1); }

output(number1 + " " + number2);
```

(a) State the values of NUMBER1 and NUMBER2 after this algorithm fragment is evaluated, given that the initial value of NUMBER1 is:

(i) 381.5

(ii) -21.0

(iii) 1200.0

(b) Complete the following algorithm for the function **abs** -

that is, write an equivalent function to the Math.abs() function.

```
double abs(double NUMBER)
{     double  ANSWER;

      . . . . . .
      . . . . . .
      . . . . . .

      return ANSWER;
}
```

7 Calculating typing fees (N96 SL P2 Q5)

In order to earn extra money, a student types extended essays for a fee. The amount charged depends on the number of pages in the document. The student charges:

- 5.00 (of some monetary unit) minimum fee for one to three pages
- 1.50 per page for each page over three pages
- an additional 3.75 if the number of pages exceeds 10.

Assuming that 200 words fit on a single typed page, a 1,300-word extended essay would produce a fee of 11.00. That is, 1300/200 = 6.5 actual pages, for which the student charges 7 whole pages. The calculation is 5.00 (for the first 3 pages) + 1.50 x 4 pages (7 - 3) to produce a fee of 11.00.

(a) Calculate the fees, showing all working, for the following extended essays of length:

(i) 1,000 words

(ii) 2,425 words.

(b) Construct the algorithm fragment that a student can use to calculate the fee.

The algorithm fragment must prompt the student to give the number of words in the extended essay. The desired output will be:

- actual number of pages
- whole number of pages
- typing fee.

Recall that **Math.floor** returns the largest (closest to positive infinity) double value that is not geater than the argument and is equal to a mathematical integer, thus **Math.floor** (769.84) returns 769.0.

The output should be clearly labelled and all variables defined.

Answers: (a)(i) 8.00 (a)(ii) 23.75

A possible solution for (b):

```
void calculateFees()
{
      int numWords, pages ;
      double typing, fee  ;

      numWords = input("How many words?");
      typing = numWords / 200;

      if ( Math.floor(typing) == typing )
      {  pages = typing; }
      else
      { pages = Math.floor(typing) + 1; }

      fee = 5;
      if (pages > 3)
```



```
            {  fee = fee +(pages - 3) * 1.5; }

            if (pages > 10)
            {  fee <-- fee + 3.75; }

            output("The actual number of pages is " + typing);
            output("Number of pages to be charged for is " +pages);
            output("The amount to be paid is " + fee);
      }
```

8 Golf scores in arrays (M97 SL P1 Q1)

The best weekly scores of golfers at a golf club are stored in two arrays as follows:

NAME		SCORE
Jenkins	[1]	82
Zendra	[2]	77
Lirmin	[3]	78
Jenkins	[4]	76
Furniss	[5]	81
Jenkins	[6]	77
	[7]	-1

The array subscripts indicate the week of the score. For example, the best score at the club in week 3 was 78, which was made by Lirmin.

Consider the algorithm below.

```
int  loc, sum, number ;
string  person ;

loc = 0;
sum = 0;
number = 0;

person = input("Enter a golfer's name");
while ((loc != 53) && (score[loc] != -1))
{  if ( name[loc].equals(person) )
   {  sum = sum + score[loc];
      number = number + 1;
   }
   loc = loc + 1;
}
answer = sum / number;
output( answer );
```

(a) State what should be the dimension of the arrays.

(b) State the sentinel value and in what array it is to be found.

(c) The variable ANSWER has not been declared. State its data type and justify your answer.

(d) Trace the algorithm with the input data "Jenkins" for the array entries given.

(e) Describe the purpose of the algorithm.

(f) Some inputs will cause the algorithm to fail. Explain when this will happen and give an outline solution to stop the error.

9 Summing loop (M97 SL P1 Q3)

The following algorithms are designed to calculate the sum (total) of a series of integers:

```
void sum1()
{     int  sum, value ;

      sum = 0;
      value = inputInt("Type an integer:");
      while ( value > 0 )
      {     sum = sum + value;
            value = inputInt("Type an integer:");
      }
      output(sum);
}

void sum2()
{     int  sum, value ;

      sum = 0;
      value = inputInt("Type an integer:");
      do
      {     sum <-- sum + value
            value = inputInt("Type an integer:");
      } while( value >= 0);
      output( sum );
}


void sum3()
{     int  sum, value, length, count ;

      length = inputInt("Type an integer:");

      sum = 0;
      for(count = 0; count < length; count = count+1)
      {     inputInt("Type an integer:");
            sum = sum + value;
      }
      output( sum );
}
```

For **each** algorithm (SUM1, SUM2 and SUM3):

(a) Explain what kind of integers (negative and/or positive) the loops are designed to accept and what effect they have on the loops.

(b) State how many times the loop is executed.

10 One-dimensional array manipulation (hashing) (M97 SL P2 Q1)

A hashing algorithm is used to place keys into a one-dimensional array, ENTRY, of length 10. All the keys are between 100 and 999 and the array location is found by using the final digit of the key. (If a clash occurs, a linear search is used to find the next available location.) For example, assuming ENTRY is initially empty (indicated by -1 in an array location) and the key to store is 447, the array would then contain:

-1	-1	-1	-1	-1	-1	-1	447	-1	-1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

(a) Draw the array after the further additions of the keys 130, 837 then 588.

(b) Assume that there is a Boolean function ATTEMPT that returns the value if the array location for the key argument is empty and false otherwise. For example, after 447 has been placed in the array above, ATTEMPT(ENTRY, 397) would return false.

The following algorithm fragment ADD is the start of a routine to add a value stored in key to the array.

```
void add(int key, int[] entry)
{  int pos;

   pos = key % 10;
   if ( attempt(entry, pos) )
   {  entry[pos] = key;
   else
   {              ...
                  ...
                  ...
   }
}
```

(i) Copy and complete ADD so that if the initial location tested in the array is not empty it will search for the next free location and add the key there. (If the end of the array is reached before a free location is found an error message is to be displayed.)

(As in all algorithms, declare any additional variables used.)

(ii) The method used in (i) is not efficient. For example, if the key 259 is to be added and there is already a key in array location 9, it will not be added, even if all array locations 0...8 are empty.

State the changes necessary to your algorithm ADD, so that it carries out the following sequence:

- search to the end of the array (unless an empty cell is found)
- go back to the start of the array and continue the search
- if no empty cell is found by the time the initial location is reached give the error message “Array totally full “.

For example, if the array has the following values:

580	441	-1	-1	-1	-1	-1	107	218	329
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

the key 438 would be added at location 2.

(iii) Compare what would happen to the contents of ENTRY in the algorithm ADD and in the calling routine when ENTRY is a pass-by-variable (reference) parameter (as given) and when it is a pass-by-value parameter.

(c) Construct the algorithm ATTEMPT for the function as used above.

(d) Give an outline solution of how the location of a stored value would be found.

A possible solution for (b) (i):

```
void add(int key, int[] entry)
   {  int pos;
      boolean found;

      pos = key % 10;
      if ( attempt(entry, pos) )
      {  entry[pos] = key;
      else
      {
          found = false;
          pos = pos + 1;
          while ((pos <= 9) && (!found))
          {
             if (attempt(entry,pos))
             { entry[pos] = key;
               found = true;
             }
             else
             { pos = pos + 1; }
          }
          if (!found)
          {  output("No free locations found"); }
      }
   }
```

A possible solution for (b) (ii):

Add a variable TRIES to count positions tried. Change while to count while tries < 10. Change the pos = pos + 1 command:

```
    int tries = 0;
    while ( (tries <= 9) && (!found)) ...
          ....
          pos = (pos + 1) % 10;
          tries = tries + 1;
```

A possible solution for (c):

```
      boolean attempt(int[] entry, int pos)
      {
            if (entry[pos] == -1)
            {  return true; }
            else
            {  return false; }
      }
```

==========

11 Finding the largest value in an array (N97 SL P1 Q1)

Consider the procedure below.

```
      void locate(int[] value)
      {  int largest, x ;

         largest = value[1];              // 1
         for( x = 2; x <= 6; x = x + 1)   // 2
            if (value[x] > largest)       // 3
            {  largest = value[x];        // 4
            }                             // 5
         }                                // 6
         output( largest );               // 7
      }
```

Copy and complete the following trace table if the data in VALUE is:

	[1]	[2]	[3]	[4]	[5]	[6]
VALUE	38	17	43	52	49	25

Line	LARGEST	x	VALUE[X] > LARGEST	**output**
1	38	-		
2		2		
3			false	

12 Binary search (N97 SL P1 Q3)

The following procedure searches for a number.

```
void search(int[] marks, int maxLength, int number)
   // Assume that the array marks contains maxLength elements
{
   int middle, upper, lower;
   boolean found;

   found = false;
   lower = 0;
   upper = maxLength + 1;
   do
   {   middle = lower + (upper - lower) / 2;
       if (number == marks[middle])
       {  found = true;  }
       else
       {  if (number < marks[middle])
          { upper = middle; }
          else
          { lower = middle; }
       }
   } while (!found);
   if (found)
   { output(middle); }
   else
   { output("Not found"); }
}
```

(a) Trace the call SEARCH(GRADES, 11, 24) by means of a trace table. Show the changes to the variables.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
GRADES	1	3	7	9	12	13	15	20	24	26	28

(b) State the output of the call SEARCH(GRADES, 7, 24).

(c) There is a problem when searching for the number 21.

(i) Identify the problem.

(ii) State what can be added to the algorithm in order to solve this problem and indicate where it must be added.

13 Simulation to count fish (N97 SL P2 Q5)

A computer is used to monitor the number of fish passing under a sensor in a lake. The sensor has a timing device that sends the character "T" to the processor every minute. If a fish is detected, the sensor sends the character "F" to the processor. At the end of the timing period, the "T" is immediately followed by an "E".

A typical data stream could be:

T F T T T F F T F F F F T F T E

This means that the first fish detected was between the first and second minute. Two fish passed under the sensor between the fourth and fifth minute, etc.

Construct an algorithm that inputs the data from the sensor and produces the following output:

- the number of minutes that the survey was in operation
- the total number of fish that passed under the sensor
- the largest number of fish that passed under the sensor in any one-minute interval.

The data that would be the output, for the above example, would be:

- the survey lasted seven minutes
- a total of eight fish passed under the sensor
- the largest number of fish in a one-minute time interval was four.

Explanation

The command that reads a character from the SENSOR into the character variable SIGNAL is:

```
signal = sensor.inputChar();
```

A possible solution:

```
void fish()
{  int fishCount, minutes, maxFish, thisFish;
   char signal;

   fishCount = 0;
   minutes = 0;
   maxFish = 0;
   thisFish = 0;

   signal = sensor.inputChar();

   while (signal != 'E')
   {
      if (signal == 'F')
      {
         fishCount = fishCount + 1;
         thisFish = thisFish + 1;
      }
      else
      {  if (signal == 'T')
```

```
            {  minutes = minutes + 1;
               if (thisFish > maxFish)
               {  maxFish = thisFish; }
               thisFish = 0;
            }
         }
         signal = sensor.inputChar();
      }
      output( "The survey lasted " + minutes + " minutes");
      output( "A total of " + fishCount + " fish passed the sensor");
      output( "The Largest number in one-minute was " + maxFish);

}
```

14 Highest values in a two-dimensional array (M96 HL P1 Q10)

The two-dimensional array of real values TEMPS outlined below contains the maximum daily temperatures over a period of one year.

DAYS -	(WEEKS)			
	[1]	[2]	...	[52]
[1]			...	.
[2]			...	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
[7]		X		

The cell indicated by the X, TEMPS (7, 2), would store the maximum temperature for Saturday in week 2.

By referring to this array, construct an algorithm that computes the highest and lowest maximum daily temperatures of the year (that is, the largest and smallest values in the array).

In addition to the temperature, the algorithm should clearly output the week of the year and the day of the week (specifying Sunday, Monday, etc) on which these extreme temperatures occurred. (Assume that day 1 is Sunday, day 2 is Monday, etc.)

Define all variables used. (You may assume that valid real temperatures are already in the array TEMPS.)

A possible solution:

```
void highLowTemps(double[][] temps)
{     int  lweek, lday, hweek, hday, x, y ;
      double  high, low ;
      high = temps[1][1];
      low = temps[1][1];
      lweek = 1;
      lday = 1;
      hweek = 1;
      hday = 1;
      for(x = 1; x <= 7; x = x + 1)
      {     for( y = 1; y <= 5; y = y + 1)
            {     if (temps[x][y] > high)
```

```
                    {     high = temps[x][y];
                          hweek = y;
                          hday = x;
                    }
                    if (temps[x][y] < low)
                    {     low = temps[x][y];
                          lweek = y;
                          lday = x;
                    }
              }
        }
        output("The highest temperature was " + high);
        output("It occurred in week " + hweek + " on a ");
        if (hday == 1)
        { output("Sunday"); }
        else if (hday == 2)
        { output("Monday"); }
        else if (hday == 3)
        { output("Tuesday"); }
              ...
              ...
        else if (hday == 7)
        { output("Saturday"); }

        output("The lowest temperature was " + low);
        output("It occurred in week " + lweek + " on a ");
        if (lday == 1)
        { output("Sunday"); }
        else if (lday == 2)
        { output("Monday"); }
        else if (lday == 3)
        { output("Tuesday"); }
              ...
              ...
        else if (lday == 7)
        { output("Saturday"); }
}
```

15 Fibonacci sequence (recursive) (M96 HL P2 Q4)

The algorithm fragment below generates values that are called the Fibonacci sequence.

```
void sequence()
{
   int p1 , p2 , x , next;
   p1 = 1;
   next = 1;
   output(p1);
   output(next);
   for(x = 1; x <= 10; x = x+1)
   {
      p2 = p1;
      p1 = next;
      next = p2 + next;
      output(next);
   }
}
```

(a) What are the Fibonacci numbers generated by this algorithm?

(b) Given the next algorithm (FIBSEQ), construct the recursive subalgorithm for the function FIBONACCI (used near the end of the algorithm) that will generate the N^{th} Fibonacci number (when N >= 2).

Define and describe all parameters and variables used in the recursive algorithm.

Hint: the n^{th} Fibonacci number [or FIBONACCI(N)] is given by the $(N - 1)^{th}$ + the $(N - 2)^{th}$ Fibonacci number, for N > 2.

```
void fibseq()
{     int  n, fib ;

      output("Enter the number, n, greater than 2,");
      output(" for the nth Fibonacci number to be generated.");
      n = inputInt("N = ");
      while (n <= 2)
      { n = inputInt("N must be greater than 2 - please re-enter:");
      }
      fib = fibonacci(n);
      output("The nth Fibonacci number is " + fib);
}
```

A possible solution for (b):

```
int fibonacci(int n)
{
   if (n == 1)
   { return 1; }
   else
   if (n == 2)
   { return 1; }
   else
   { return fibonacci(n - 1) + fibonacci(n - 2); }

}
```

16 Binary search (N96 HL P2 Q1)

(a) To solve a particular computer problem, data has to be stored in sorted order.

If an array is used, a sorted order can be obtained as the data is entered initially, for example, if the input data is:

Grapefruit, Apple, Banana, Pear, Orange, Grape, Kiwi.

If the array to store them is fruit, after the first read it would store:

Grapefruit			

After the second read, it would store:

Apple	Grapefruit		

(i) Draw the array to show how it would look after every subsequent read.

(ii) Draw and describe a data structure that would not require as much data rearrangement, when a new item is added, to maintain a sorted order. Give brief details of its organization.

(b) A binary search can be used to locate a data item in a sorted array of size N. It does this by comparing the middle item of a list with the required input value. If the input is equal to the middle item of

the list the search stops, otherwise the half of the list that cannot contain the item is ignored by reassigning the current value for the top or bottom of the list.

For example, using the final array from (a)(i) above, where N= 7 and the item to find is Kiwi :

initially bottom is 1 and top is 7 (that is, N).

Apple	Banana	Grape	Grapefruit	Kiwi	Orange	Pear

Thus middle = (top + bottom) / 2 = 4

Since the item at location 4 (Grapefruit) is not equal to the input value (Kiwi) the search does not stop and since Grapefruit is less than Kiwi, the input value cannot be in the lower half of the list, so bottom is reassigned to middle + 1 and the search continues.

(i) Continue the trace of the binary search, using a suitable layout.

(ii) Construct the algorithm fragment to perform the binary search. (Remember to give an error message if the item is not found.)

(c) The binary search can be expressed as a recursive routine.

(i) Explain the term *recursion* and state one necessary condition required by any recursive routine.

(ii) A call to a recursive binary search routine when looking in array fruit for item seek could be:

bsearch(seek,bottom,top,fruit).

Construct the recursive algorithm fragment that would carry out this call. No loop is used, instead the routine is used recursively with suitable parameters.

A possible solution for (b) (ii):

```
void binSearch(String seek, int bottom, int top, String[] fruit)
{
     //  looking for the occurrence of seek within fruit

     int  middle ;

      do
      {     middle = (bottom + top) / 2;
            if (fruit[middle].equals(seek))
            {  output("found"); }
            else
            {     if (fruit[middle].compareTo(seek) > 0)
                  {     top = middle - 1;  }
                  else
                  {     bottom = middle + 1; }
            }
      } while ( !fruit[middle].equals(seek) && (top >= bottom) );

      if (top < bottom)
      {     output("error"); }
}
```

A possible solution for (c) (ii):

```
void bSearch(String seek, int start, int finish, String[] fruit)
{
```

```
  //  Looking for the occurrence of SEEK within FRUIT recursively

      int  middle ;
      if (finish < start)
      {  output("error"); }
      else
      {     middle = (start + finish) / 2;
            if (fruit[middle].equals(seek))
            {  output("found"); }
            else
            {     if (seek.compareTo(fruit[middle]) < 0)
                  {     bSearch(seek, start, middle - 1, fruit); }
                  else
                  {     bSearch(seek, middle + 1, finish, fruit); }
            }
      }
}
```

17 Binary tree (N96 HL P2 Q7)

A binary tree is stored in a dynamic data structure as follows. Each node is given in the format:

Address
node_entry
left_link
right_link

350		380		193
+		/		B
327		350		-1
419		257		-1

327		419		287		257
A		*		C		D
-1		193		-1		-1
-1		287		-1		-1

(a) State what the values associated with left_link and right_link represent.

(b) Draw the binary tree presented above if the root node is 380.

(c) A recursive routine is used to traverse this tree. Assume the new type NODE has been defined as given below.

```
class Node
{
   char nodeEntry ;
   Node leftLink ;
   Node rightLink ;
}

void traverse(Node value)
{
```

```
   if ( value.leftLink != null)
   {
      traverse(value.leftLink);
   }
   if ( value.rightLink != null)
   {
      traverse(value.rightLink);
   }
   output( value.nodeEntry );
}
```

Trace the algorithm with the initial call TRAVERSE(380).

(d) Give the in-order traversal of the tree given in (b).

(e) State the changes needed to be made to the algorithm in (c) to generate an in-order traversal of a tree.

18 Summing in a two-dimensional array (N97 HL P1 Q10)

The two-dimensional array, called grid, has the following values.

8	16	12	19	4
3	7	15	21	1
2	14	17	5	10
6	13	12	18	9

For example, grid[3, 2] = 14.

Consider the algorithm fragment below:

```
for(row = 1; row <= 4; row = row+1)          // 1
{   for(col = 2; col <= 5; col = col + 1)    // 2
    { if (grid[row][col] < grid[row][col-1])// 3
      {   grid[row][col] = grid[row][col-1];// 4
      }                                      // 5
   }                                         // 6
}                                            // 7
```

(a) Redraw the array and its new contents **after** tracing the algorithm above.

(b) Construct the algorithm that displays both the average (mean value) of each column and the overall average.

A possible solution for (b):

```
void average(int[][] grid)
{
   int row, col, colsum, allsum ;
   double colavg, allavg ;

   allsum = 0;
   for(col = 1; col <= 5; col = col + 1)
   {  colsum = 0;
      for(row = 1; row <= 4; row = row + 1)
      {  colsum = colsum + grid[row][col];
         allsum = allsum + grid[row][col];
      }
      colavg = colsum / 4 ;
      output("In column " + col + " the average is " + colavg);
   }
```

```
   allavg = allsum / 20 ;
   output("The overall average is " + allavg);
}
```

19 Random numbers (N97 HL P2 Q1)

One method of generating pseudo-random numbers is to start with two values (seeds), multiply them together and then extract the middle digits. This value can then be used as a seed to generate the next value.

For example, if 85 and 65 are used as the seeds, the following is obtained:

Iteration	Seed 1	Seed 2	Product	Pseudo-random number
1	85	65	5525	52
2	65	52	3380	38
3	52	-	-	-

(a) Complete the table above to show up to iteration 6.

(b) Construct the algorithm pseud that will calculate and display the first six pseudo-random numbers. You may assume that there is a function extract that returns the middle two digits from a four-digit number. For example, extract(5525) would return 52.

(c) Construct the algorithm for extract as used in part (b).

Recall that **Math.floor** returns the largest (closest to positive infinity) double value that is not greater than the argument and is equal to a mathematical integer, thus **Math.floor** (43.8) returns 43.0.

(d) Explain what would happen if the product was 2008.

(e) Another way to generate pseudo-random numbers is to use modulo arithmetic. For example, in a sequence of values the new number is calculated by multiplying the last generated random value by a set number and using the remainder after dividing by another set value.

For example, with an initial seed of 4, a multiplier of 3, and a modulus of 7, the following is obtained:

Iteration	Pseudo-random number (R_n)	$3*R_n$	$R_{n+1} = (3*R_n) \% 7$
1	4	12	5
2	5	15	1
3	1	3	3

(i) Construct the recursive algorithm pseud2(iter**,** num) that outputs any value in the sequence. For example, the call pseud2(2, 4) would output 5, where the first parameter represents the sequence number required and the second parameter represents the seed.

(ii) Construct pseud2 as an iterative, rather than a recursive, algorithm.

(iii) State **one** advantage and **one** disadvantage of recursive routines when compared to iterative routines.

A possible solution for (b):

```
void pseud()
{  int  seed1, seed2, x, product, random ;

   seed1 = 85;
   seed2 = 65;
   for(x = 1; x <= 6; x = x + 1)
   {  product = seed1 * seed2;
      random = extract(product);
      output( random );
      seed1 = seed2;
      seed2 = random;
   }
}
```

A possible solution for (c):

```
int extract(int value)
{
   int  hun, left ;

   value = value / 10;
   hun = value / 100;
   hun = hun * 100;
   left = value - hun;
   return left;
}
```

A possible solution for (e) (i):

```
void pseud2(int iter,int num )
{  if (iter == 1)
   {   output( num ); }
   else
   {   pseud2(iter - 1, (3 * num) % 7 ); }
}
```

A possible solution for (e) (ii):

```
void pseud2(int iter ,int num )
{  int res, times ;

   res = num;
   if (iter > 1)
   {  for(times = 2; times <= iter; times = times + 1)
      {  res = (res * 3) % 7; }
   }
   output( res );
}
```

Teaching order

This section shows two types of teaching order for computer science courses at SL and HL. The aim is to help teachers work out the best timetable of lessons for their individual situations, while recognizing that there are many different ways to teach these courses. The suggested teaching orders have been written by current computer science teachers, who are also DP examiners.

Teaching order 1

The order of topics in the *Computer Science* guide is not a suitable teaching order. For example, teaching topic 1 effectively requires previous knowledge of several subtopics in topics 2 and 3.

A conventional approach to teaching computer science starts with hardware and covers architecture, peripheral devices, backing store, languages, operating systems, networks and, finally, any recent developments. When teaching topic 2—programming—the teacher needs to decide when to introduce object-oriented concepts for HL students.

This approach is well established and results in a teaching order similar to the schemes shown overleaf. This assumes the academic year is split into four equal parts; some adjustment would be necessary for other year/term structures.

Possible SL outline

Term	Mainly theory	Topic	Java and program design	Topic
1	Computing system fundamentals—1	3.1 3.2	Learning Java—basics: data members, applets, simple data types, simple constructs, programming style	2.1 1.6
2	Computing system fundamentals—2 Networks	3.3 3.4	Sequence, selection and repetition in depth Methods and parameters Mini project—usability issues	2.1 1.3
3	Computer systems	3.5 3.6 3.7	Classes, applications, OOP/design/ programming concepts, testing solutions Mini-project, design and test	1.5
4	Review Systems life cycle	3 1.1 1.2	Files Preparation for dossier project	2.1 Dossier
5	Algorithms	2.1	Dossier work	Dossier criteria
6	The case study	1.4	Dossier work	Dossier criteria
7	Algorithm practice	2.1	Algorithm practice	2.1
8	Examination		Examination	

Possible HL outline

Term	Mainly theory	Topic	Java and program design	Topic
1	Computing system fundamentals	3.1 3.2 3.3 3.4	Learning Java—basics: data members, applets, primitives, constructs, programming style	2.1.6
2	Computing systems and further fundamentals Networks	3.5 3.6 3.7 6	Methods and parameters, classes, applications, event-driven and OOP programming concepts Mini project—usability issues	2.1 1.3
3	Computer mathematics and logic	4	Arrays, records, testing solutions Mini-project, design and test	1.5
4	Algorithms and data structures (stacks, queues, lists) File organization Systems life cycle	5 7 1.1 1.2	Files, ADTs, recursion Preparation for dossier project	2.1 5 Dossier
5	Algorithms	2 5	Dossier work	Dossier criteria
6	The case study	1.4	Dossier work	Dossier criteria
7	Algorithm practice	2.1	Algorithm practice	2.1
8	Examination		Examination	

Some adjustments can be made to this teaching order depending on the way classes and lessons are set up in individual schools. For example, if the group combines SL and HL students (which is **not** recommended), then it may be preferable to teach core subjects to the group as a whole. In this case, the HL students usually have more lessons per week than the SL students. Two further approaches are possible here.

1. Teach core subjects in the joint lessons and HL topics in the extra lessons, following a pattern of, for example, four joint and two HL-only lessons per week throughout the year.

2. Teach the combined group of SL and HL students for six periods per week during terms 1 and 2. Then teach HL students only for term 3. Bring back the SL students for four periods per week in term 4.

You will need to make adjustments to suit your particular school. The success of the teaching order depends on many factors, including the flexibility of administrators.

Teaching order 2

Combined SL and HL group.

Teaching time: 5 x 45 minutes per week over 62 weeks.

The students start with no programming skills. Some adjustments can be made to the teaching order depending on students' previous knowledge. For stronger students and those with programming experience, GUI programming can be introduced instead of simple input/output (IBIO).

It is important to give students regular practice of examination questions using past examination papers.

SL students do not attend all the lessons.

Year 1

WEEK	COMMON CORE—SL & HL	ADDITIONAL HL
1.	General computer science topics - Introduction to computer systems - Analog and binary signals - Computer memory - The processor - Input devices - Output devices - Storage devices	Magnetic disk storage
2.	Software Machine language and high-level languages Language translation and interpretation Test and analysis (computer basics)	Functions of an operating system - Linker - Loader - Library manager
3.	Using an IDE How to edit, compile and run example Java programs Navigating the Java directories	
4.	Primitive data types Variables Assignment statements Type casting Arithmetic operators and precedence rules Simple output Programming exercises	

WEEK	COMMON CORE—SL & HL	ADDITIONAL HL
5.	Simple input and output String constants and variables Concatenation of strings String methods Reading input Formatting output Programming exercises	
6.	Test and analysis (tracing and constructing simple sequence programs) Boolean expressions Relational operators The single-branch **if** statement Programming exercises	
7.	Nested statements Multi-branch **if-else** statements Programming exercises	
8.	The **switch** statement Programming exercises Test and analysis (tracing and constructing programs—**if, switch**)	
9.	Loops The **while** statement Counting loops Sentinel-controlled loops Programming exercises	
10.	File input and output Reading data from the file Software life cycle	
11.	Designing a program following the major stages of the software life cycle	
12.	Test and analysis (tracing and constructing programs—iterative algorithms) Review	

WEEK	COMMON CORE—SL & HL	ADDITIONAL HL
13.	Class and method definition Variables in class definition Using methods Methods that return a value Parameters—how argument values are passed to a method	
14.	Defining class methods Accessing data members in a method The variable **this** Initializing data members Programming exercises	
15.	Constructors The default constructor Creating objects of a class Passing objects to a method The lifetime of an object	
16	Class design example	Recursion
17.	- Statement of the problem - Analysis of the problem - Software design - Program construction - Testing and debugging - Installation and operation - Maintenance	
18.	Revision	
19.	WRITTEN EXAMINATION—END OF THE FIRST SEMESTER	
20.		Features of an object Encapsulation Data hiding Polymorphism Inheritance
21.	Programming exercises	Tracing and constructing algorithms

WEEK	COMMON CORE—SL & HL	ADDITIONAL HL
22.	The **for** statement The **do** statement Nested statements Programming exercises	
23.	One-dimensional array - Array variables - Defining an array - Accessing array elements - Initializing arrays - Array length - Common array algorithms	
24.	Linear search Insertion Deletion Programming exercises	
25.	Test and analysis (arrays) Selection sort Efficiency of selection sort Programming exercises	Merging two sorted arrays into the third array
26.	Bubble sort Efficiency of bubble sort Programming exercises Two-dimensional (2-D) arrays	
27.	An array example—dividing program into classes	
28.	Storing objects Sorting objects Programming exercises	
29.	Ordered and unordered arrays Binary search Efficiency of binary search	Recursive binary search Recursion versus iteration
30.	Test and analysis (sorting and searching)	Quicksort Comparing the simple sorts Comparing the searching algorithms BigO notation

WEEK	COMMON CORE—SL & HL	ADDITIONAL HL
31.		Stack definition Tracing and constructing algorithms that implement a stack in an array
32.		Infix, prefix, postfix notation Usage of stacks Test and analysis (stacks, algorithm efficiency)
33.		Queue definition Tracing and constructing algorithms that implement a queue in an array Usage of queues
34.		Hash tables Handling clashes Programming exercises
35.	WRITTEN EXAMINATION—END OF YEAR 1	

Year 2

WEEK	COMMON CORE—SL & HL	ADDITIONAL HL
36.	Systems life cycle - Stages - Systems analysis - Systems flowcharts - Data security and integrity - Documentation	
37.		Linked structures (single and multi-linked structures) - Terminology, definitions - Logical representation (diagrams)
38.		Simply linked lists - The linked list class - Display - Search - Insert - Delete Programming exercises
39.		Stack—linked list implementation Queue—linked list implementation Test and analysis (linked structures)
40.		Binary trees - Terminology - Binary search trees - Tree traversal - Drawing diagrams showing how to insert and delete nodes - Trees represented as arrays
41.		Binary trees—programming exercises - Tree class - Find - Insert - Delete - Display
42.		Files—programming exercises - Reading/writing binary files - Searching - Inserting directly into file - Deleting directly from file - Merging
43.		File organization Test and analysis (files, trees)

WEEK	COMMON CORE—SL & HL	ADDITIONAL HL
44.	DOSSIER WORK	
45.		
46.	Computer architecture - CPU - Primary and secondary memory - Buses - Registers - Fetch-execute cycle	Processor configuration - ACC, IR, PC - Interrupts
47.	Networked computer systems - Network topologies - Hardware and software required in networking - Data integrity and security - Applications and implications	Networked computer systems - WAN - Protocol - Communications - Security
48.	Operating systems - Functions - Single-tasking and multitasking environments - Methods of processing Utility software Test and analysis (networks and OS)	Computer/peripheral communication - Peripheral control - Polling - Interrupts
49.	Number systems - Binary - Decimal - Octal - Hexadecimal Data representation - Use of binary to represent data - Method-of-two's complement integer representation	Binary and hexadecimal calculations
50.		Real number representation - Mantissa, exponent, normalization - Floating-point representation - Fixed-point representation - Errors
51.	DOSSIER WORK	
52.	WRITTEN EXAMINATION—END OF THE THIRD SEMESTER	

WEEK	COMMON CORE—SL & HL	ADDITIONAL HL
53		Boolean logic - Expressions - Truth tables - Simplification - Logic gates - Logic circuits
54.	DOSSIER WORK	
55.		
56.	Social significance and implications of computer systems Case study	
57.		Case study—HL topics Discussions, some review notes
58.	MOCK EXAMINATION	
59.	Revision Examination preparation exercises	
60.		
61.		
62.		
63	IB EXAMINATION	

Examples of student work, highlighting IA criterion levels

Introduction

These examples of student work show how teachers have applied the assessment criteria. The examples are not intended to demonstrate the "correct" way of approaching any particular section of the program dossier. In particular, they are not examples or templates for students to use and follow.

The section lengths are merely a guide. They will vary from problem to problem, and according to the students' ability to express themselves concisely.

The sample material presented here is either taken from dossiers completed under the old course and criteria or, in some cases, prepared by examiners. The examples that are actual student work are presented in their original styles, which may include spelling, grammatical and any other errors. Since the examples may have been edited or altered for the purposes of illustrating the present criteria, the levels awarded may vary from those awarded to the students when the materials were "live".

Stage A—Analysis

Criterion A1: Analysing the problem

The documentation should be completed first and contain a thorough discussion of the problem that is being solved. This should concentrate on the **problem** and the goals that are being set, not on the method of solution. A good analysis includes information such as sample data, information and requests from the **identified end-user**, and possibly some background of how the problem has been solved in the past.

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below. For example, the student has simply described the programmed solution.
1	The student only **states** the problem to be solved **or shows** some evidence that relevant information has been collected.
2	The student **describes** the problem to be solved.
3	The student describes the problem **and provides evidence** that information relating to the problem has been collected.

This section of the program dossier would typically be two to three pages in length. It should include a brief statement of the problem as seen by the end-user. A discussion of the problem from the end-user's point of view should take place, including the user's needs, required input and required output. For example, evidence could be sample data, interviews and so on, and could be placed in an appendix.

Level 1

Achievement level	Descriptor
1	The student only **states** the problem to be solved **or shows** some evidence that relevant information has been collected.

The following is an example of a level 1 analysis.

Statement and Analysis of Problem
Every business has to keep a record of their expenditures and other transactions. This program is attempting to solve the problems that many businesses are running into with manual bookkeeping by creating a employee planning system. A payroll is the most important aspect of any business, so it is imperative that the records are kept organized and the employer knows how much money will be needed to pay its employees at the end of each month. There are many restrictions in keeping bulky hand-written records such as the space needed to store them and the time it takes to organize all of these files. This problem is magnified for small business that are trying to expand, for it is difficult to them continually allocate storage space in an office for

manual filing while, simultaneously, trying to provide more space for new employees. This program WM provide a way for businesses to keep accurate records of the current wages paid to their employees on an individual basis, making the data readily available and presents it in a way that is easy to modify. Information about each employee will be easy to change, eliminating problems with records being confused due to name or other changes.

The user will start by selecting an option from the menu and the program will execute that option. Each option will most likely be a function. With this program the user would have the ability to create a record for an employee with the following fields-name (last, first), social security number, address, phone number, starting date and current hourly salary. Each object is stored in a file at a certain address specific to the employee, denoted by a corresponding "employee number". Individual employee files will be able to be recalled by employee number, returning the name, address, social security number, phone number, starting date and hourly salary; the information in field can be changed at any time.

The user will have the option of viewing a planner that will provide the user with the amount of money used to pay all the employees, assuming all are working full time, for the average month. This is the planning aspect of the program. A company's main objective is to provide goods and services to be sold to the private sector at a price. For virtually every industry this would not be possible if there were no employees to work. In this sense it is essential for an employer to know much money it will need to pay its employees at the end of a month in order to make decisions for future expenditures. Without this knowledge the company could experience dire financial problems and even go bankrupt.

The user will also have the option of viewing an alphabetized list of its employees - presenting the data input in the system in an organized manner. Choosing this command will return a list of the entire database of names in the file alphabetized from A-Z by last name. This allows the user access to an organized reference, which can be used for mailing lists and other functions that need to be performed by the companies. The user will also be able to easily remove an employee from their records, which saves money and storage, space.

This program will rely heavily on modularity to execute each of the menu options. Mastery of each of the seven aspects will be seen through each of the options

Students who complete the analysis after the program has been written will probably fall into the trap of describing the solution rather than the problem.

Level 2

Achievement level	Descriptor
2	The student **describes** the problem to be solved.

The following example illustrates this achievement level.

Statement of the Problem and Analysis

How could a hotel company keep track of rented and reserved rooms, the bill for each room, and their characteristics (ex: smoking or non-smoking)? For this to be done, a program is need that would organize all the rooms so that they could be accessed by characteristics or room number, would be able to mark rooms as either vacant or not, and reserve rooms for a renter.

This problem has been solved in a variety of ways by different hotel companies. Some companies enable a customer to reserve a room with the credit card number while others only require a name. Other differences individual hotels have are evident in their procedures for collecting money. Customers are either required to pay up front or have the choice of paying when they check-out. To resolve this problem the program would need two different types of input from the user. First of all, it would have to let the user create a hotel file. The input would have to include the number of floors and the rooms per floor of the hotel. The initial set-up of the hotel would also require input of information about the price per room, services offered, and the characteristics of each room. With this information, a linked list of floors could be created. Each node of the floor list would then contain a linked list of individual room structures. The second type of input a user would give would be information to reserve a room, purchase room service, or pay the room bill. When a room is rented, the dates needed and the person's name would be required to mark the room as reserved. When a service is purchased, the room bill will increase by the set price of whatever is ordered. A customer would be able to pay the bill at any time during their stay. The only input needed would be the room number. Lastly, the desk-receptionist would be able to gain access to the any of the characteristics of a certain room by entering the room number.

While this is not a great improvement on the previous example the student does describe previous attempts to solve the problem and also describes the input data required. It also appears that the student has at least visited a hotel and observed how it works. This student additionally supplied a detailed table of inputs and outputs (see below) to accompany this description, and this helped to confirm the award of level 2.

What program inputs	Who provides it	Frequency	Acquisition cost	Used where
Number of floors	Hotel creator	1		To build a simulated hotel
Rooms per floor	Hotel creator	1		To build simulated hotel
Room rate	Hotel creator	1 (but could be changed)		To calculate the money made
Reservations (date)	Desk person	Lots	Hourly wage	To reserve the room
Reservations (name)	Desk person	Lots	Hourly wage	Identification
Furniture and characteristics of each room	Hotel creator	1 time per room		To tell the customers about the room
Available services and their costs	Hotel creator	1		To tell the customer about the available services
Services used by customer	Desk person	Lots	Hourly wage	To keep track of how much the customer owes
Room number	Desk person	Lots	Hourly wage	To check on the status of the room

There is no evidence that the student has collected any information. If the student did indeed visit a hotel, they might have collected a copy of the registration form, a print-out of a bill and perhaps a screenshot of the existing system.

For the same reason the following example also represents achievement level 2.

Problem to be solved:

Ever since role-playing games have become popular, more and more character profiles and statistics have to be recorded. An example of a typical character's statistics (for the intended program based on the Witchcraft system) is as follows.

Name: Brett Compton
Age: 24
Sex: Male
History: Not Given
Archetype: Reluctant hero
" " *Description*: Not Given
Character Type: Gifted
Covenant: Wicce

Attributes

Name	Level
Strength	2
Dexterity	3
Constitution	4
Intelligence	4
Perception	4
Willpower	3
Life Points	34
Endurance Points	29
Speed	14
Essence Points	20

Skills:

Name	Level
Computers	3
Programming	4
Driving (car)	3
First Aid	2
Myth and Legend	2
Occult Knowledge	2
Research/Investigation	1
Humanities (Wicce Theology)	2
Rituals (Wicce)	3
Science (Astronomy)	2
Science (Physics)	2
Swimming	2
Trance	4

Metaphysical Skills:

Name	Level
Blessing	3
Insight	3
Shielding	3

It is hard to keep track of so much information at a time. Many times, character's sheets with this information are often lost—either by a careless player or a game master. I chose to solve this problem by creating a program that will create a character record filing system for players/game masters specifically for this Witchcraft system.

With the role-playing boom, the creation of programs to handle character creation has become more common. There is a plethora available on the Internet, ready to download, but they are only limited to the mainstream role-playing games such as Advanced Dungeons and Dragons or World of Darkness or Star Wars, etc.

I found that it is impossible to find a character generator for almost nameless role-playing games. I figured that the other Witchcraft players on the Internet could find this program useful if I choose to share it.

The character generation in the Witchcraft system has some basis of calculation so this program can limit problems that Game Masters may face with having to calculate secondary attributes for the characters.

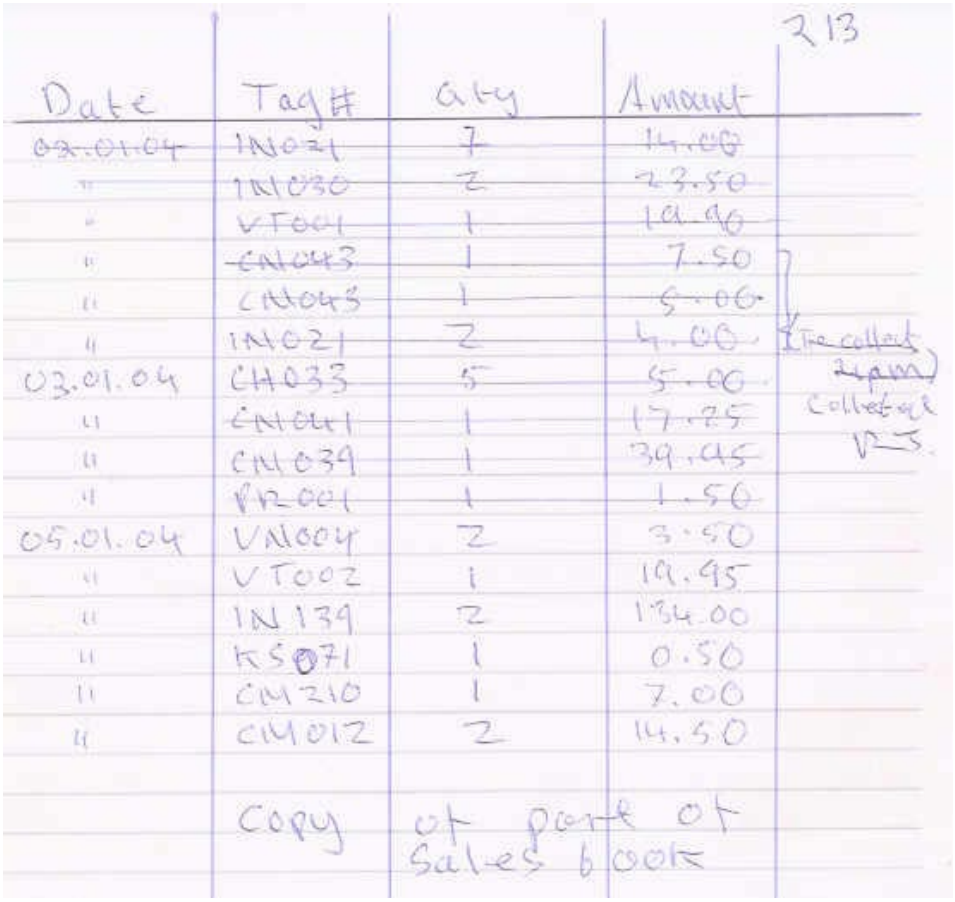
```
The program will calculate it itself to reduce strain on the GM's and
prevent miscalculation or cheaters from adding too many points (with
the ruse that it was a miscalculation).

Other character generation programs that I've seen have a point
allocation interface that became obsolete when it came to manipulating
statistics. I have often searched the Internet for a version that
could create new character files, and also change the character's
statistics as the character develops. I decided that after finishing
this, I could code a second version of the program that that will
allow the program to be able to modify the data inside the file and
update the character statistics.
```

This example also omits details of the processing required and how the outputs will be produced.

Level 3

Achievement level	Descriptor
3	The student describes the problem **and provides evidence** that information relating to the problem has been collected.

The collection of evidence can be problematic. The user may not be willing to give copies of important documents to a student, or they may contain confidential information. In such cases it would be acceptable for the student to present information copied into a notebook.



213

Date	Tag #	qty	Amount	
~~02.01.04~~	~~IN021~~	~~7~~	~~14.00~~	
"	~~IN030~~	~~2~~	~~23.50~~	
"	~~VT004~~	~~1~~	~~19.95~~	
"	~~CN043~~	~~1~~	7.50	
"	~~CN045~~	~~1~~	~~5.00~~	
"	~~IN021~~	~~2~~	~~4.00~~	The collect 24pm Collected VT5
03.01.04	CH033	5	5.00	
"	~~CN041~~	~~1~~	~~17.25~~	
"	CN039	1	~~39.45~~	
"	~~PR004~~	~~1~~	~~1.50~~	
05.01.04	VN004	2	3.50	
"	VT002	1	19.95	
"	IN139	2	134.00	
"	KS071	1	0.50	
"	CM210	1	7.00	
"	CM012	2	14.50	
	Copy	of part of Sales book		

Other forms of evidence might include the following (see the subject guide, subtopic 1.2.2).

- Transcripts of interviews
- Copies of questionnaires with a tabulated summary responses
- Photographs or sketches
- Descriptions of previous attempts to solve the problem by experts in the field

Criterion A2: Criteria for success

This section of the program dossier will clearly state the objectives/goals of the solution to the problem.

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student **states some** objectives of the solution.
2	The student **describes most of** the objectives of the solution.
3	The student **relates all of** the objectives of the solution to the analysis of the problem.

This section of the program dossier would typically be one to two pages in length. Objectives should include minimum performance and usability. These criteria for success will be referred to in subsequent criteria, for example C3 (success of the program) and D2 (evaluating solutions).

Level 0

Achievement level	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.

This level could be awarded to a student who merely describes or outlines the programming solution after the work has been completed.

Level 1

Achievement level	Descriptor
1	The student **states some** objectives of the solution.

The following section outlines the problem and then goes on to state some objectives of the solution (these have been **bolded** for clarity).

In many cases it might be more appropriate to present the objectives as a clear series of bullet points.

Problem Analysis

The problem I have chosen is a vaccination database of a Vet. The system currently used is a file system with cards, where all the details of the treated animals are stored on. For every vaccination there has to be made another card, which is then put into the binder of the specific animal. Each storage binder shows the name, animal type, gender, date of birth, owner, owner's address and the Vet's name. The Vaccination certificates put inside contain the type of the vaccine, the vaccine number, the date of the vaccination and the date where the next vaccination is due to. The binders are stored in separate groups

for animals, which are sorted by names of the owners in alphabetical order. These Binders are very inconvenient in terms of handling and storing data, because the single vaccination certificates might get lost or, if the owner has got more than one animal in treatment, the binders can get easily confused. Additionally the binders take up a lot of space. For each vaccination there must be a new record created.
A database system would be much more effective in handling and storing this data. Because in every room where animals get treated, the Vet can get access to the database via a computer with the database software installed. He can then ***add a new vaccination record as soon as the vaccination is given***. Also ***special information about this particular animal might be displayed before the treatment is chosen***. This vaccination record is then added to the animal's details and ***can be accessed from every computer in the network*** with this software installed. Other errors are excluded by having a special format for the data put in. For example the date has to be put in with the format dd/mm/yyyy. ***If an animal eventually dies the records can obviously be deleted easily***.

Level 2

Achievement level	Descriptor
2	The student **describes most of** the objectives of the solution.

The following example describes many of the objectives of the solution without relating them to the analysis section. This level might be a suitable maximum for a student who has not actually analysed the problem at all.

Criteria for success

The server program will allow for the modification of questions that the program will ask. This will be used by the teacher to add and save sets of questions that can be loaded and saved to be used for future classes. So, for the end user the process would go as follows. The teacher would spend some time entering questions into the data entry portion of the program. These would simply be composed of a question, an answer, and a category the question falls into. From there, the teacher could run the server portion of the program which would allow for students to join the game and participate in answering questions which would in turn give him points for each question like a typical jeopardy game. If they try to answer the question by buzzing in they will gain the number of points if it is correct and lose the number of points if they answer it is incorrect. When a user opts to answer a question by buzzing in he will be forced to choose from several possible answers which will be randomly generated from other questions within the category. The user who correctly answers the question will have the option of choosing another problem from the matrix of questions where each column has questions that fall into a certain category and each row has questions of a certain point value.

Level 3

Achievement level	Descriptor
3	The student **relates all of** the objectives of the solution to the analysis of the problem.

This level can only be applied by considering the analysis and the criteria for success together. There must be a clear connection between the preceding analysis and the criteria or goals that are set in this section. In the following example, connections to the analysis section have been bolded for clarity.

A2 – Criteria for Success
The testing system:
This is described in the analysis under **Figure 2.**

Plays a set piece of music during the test – **see Figure 1.**

As **required by the analysis** the testing system must collect the relevant information from the test taker, this is:

- name
- sex
- answers to maths questions - numeric
- answers to music questions – true/false;

Calculates:

- the score
- the time taken

Stores all of the above data in a csv file **as described in the analysis section.**

The user controls
The user may:

- Edit the introduction (Figure 5)
- Edit the numeric questions (Figure 4)
- Edit the music questions (Figure 4)
- Edit the piece of music to be played (Figure 6)
- View/Access/Export the results file (Figure 7)

The criteria for success will be examined further in criterion C3 (success of the program).

Criterion A3: Prototype solution

The prototype solution **must** be preceded by an initial design for some of the main objectives that were determined to be the criteria for success. A prototype of the solution should be created.

A prototype is: "The construction of a simple version of the solution that is used as part of the design process to demonstrate how the system will work."

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student includes an initial design **and a prototype**, but they do not correspond.
2	The student includes an initial design and a prototype that **corresponds**.
3	The student includes an initial design and a complete prototype that corresponds to it **and documents user feedback** in evaluating the prototype.

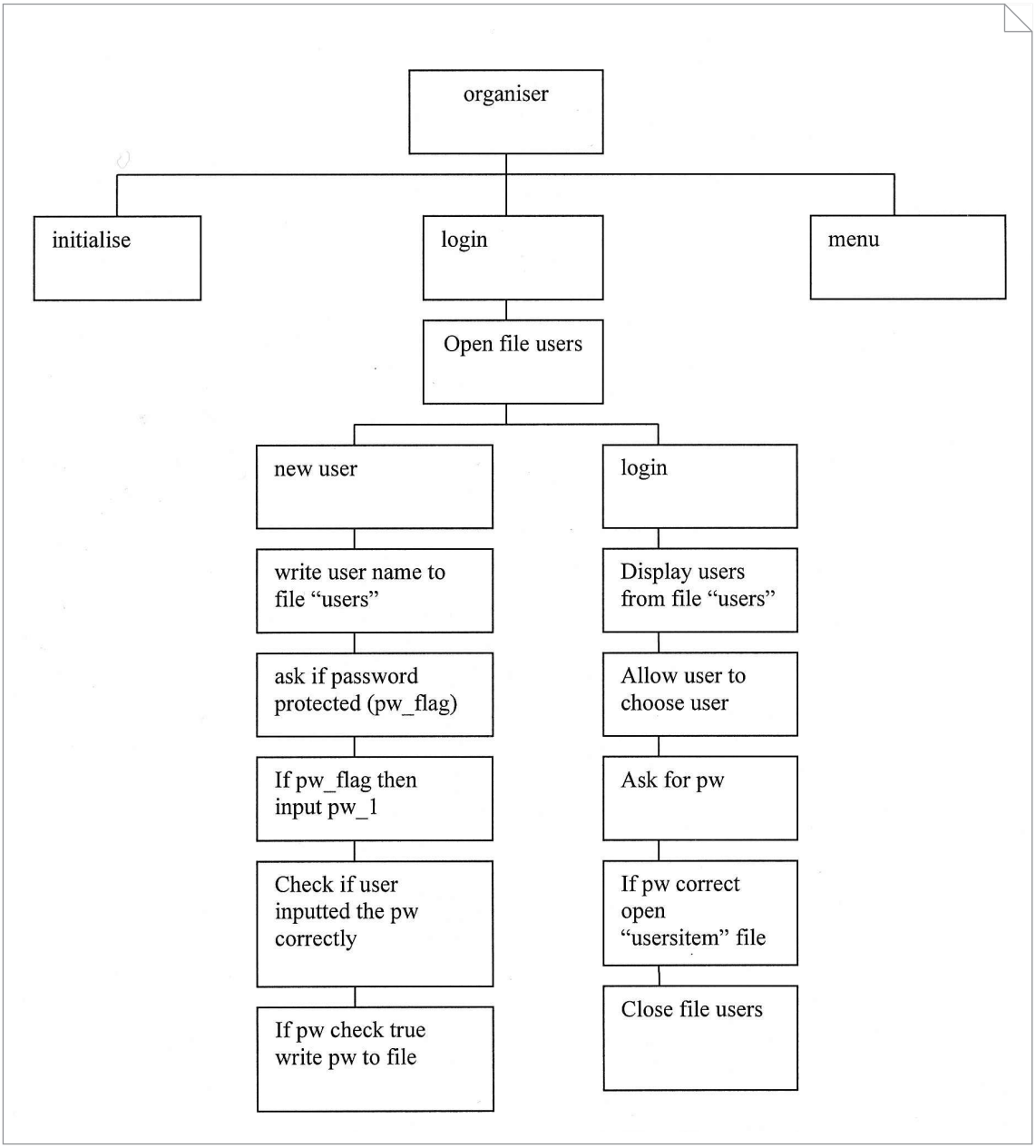
The prototype need not be functional, it could be constructed using a number of tools such as: Visual Basic, PowerPoint, Mac Paint, Corel Draw for a simple Java program. The intent is to show the user how the system is expected to operate, what inputs are required and what outputs will be produced. A number of screenshots will be required for the user to be able to evaluate the solution properly. The prototype, at its simplest, could be a series of clear, computer-generated drawings, a hierarchical outline of features in text mode, or a series of screenshots.

Documentation of user feedback could be, for example, a report of the user's comments on the prototype.

Level 1

Achievement level	Descriptor
1	The student includes an initial design **and a prototype**, but they do not correspond.

This is an example of an initial design.



The prototypes presented may take various forms. The following is an example of a text-based prototype.

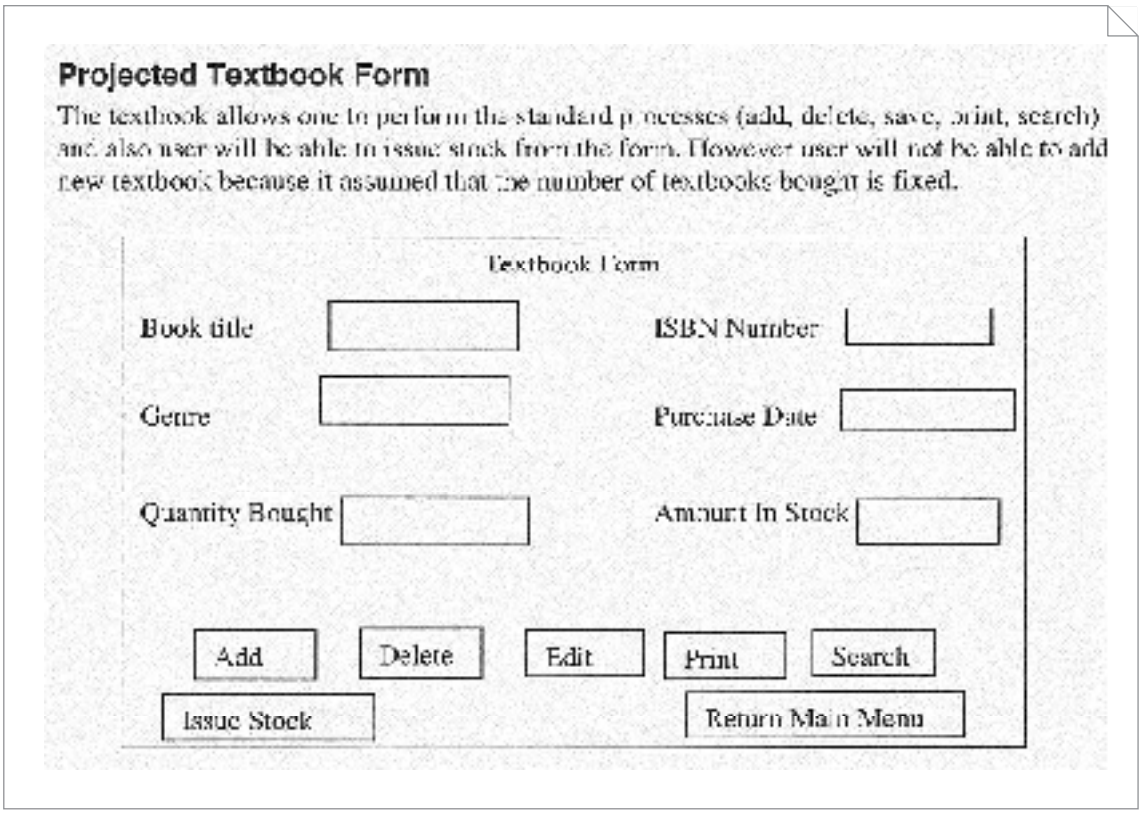
```
1.1 Search Student Flight Details

ID number: 89
First name: Nat
Last name: Gullayanon
Boarding house: Kamala
Year: 13
Flight number: TG 345
Origin: Phuket
Destination: Bangkok
Departure date: 23/7/2004
Departure time: 12:00 AM
Arrival time: 12:00 PM

          Press Enter to continue...
```

Diagram 5: This screen will show all the flight details of the record that contain ID number of 89. (P1.1)

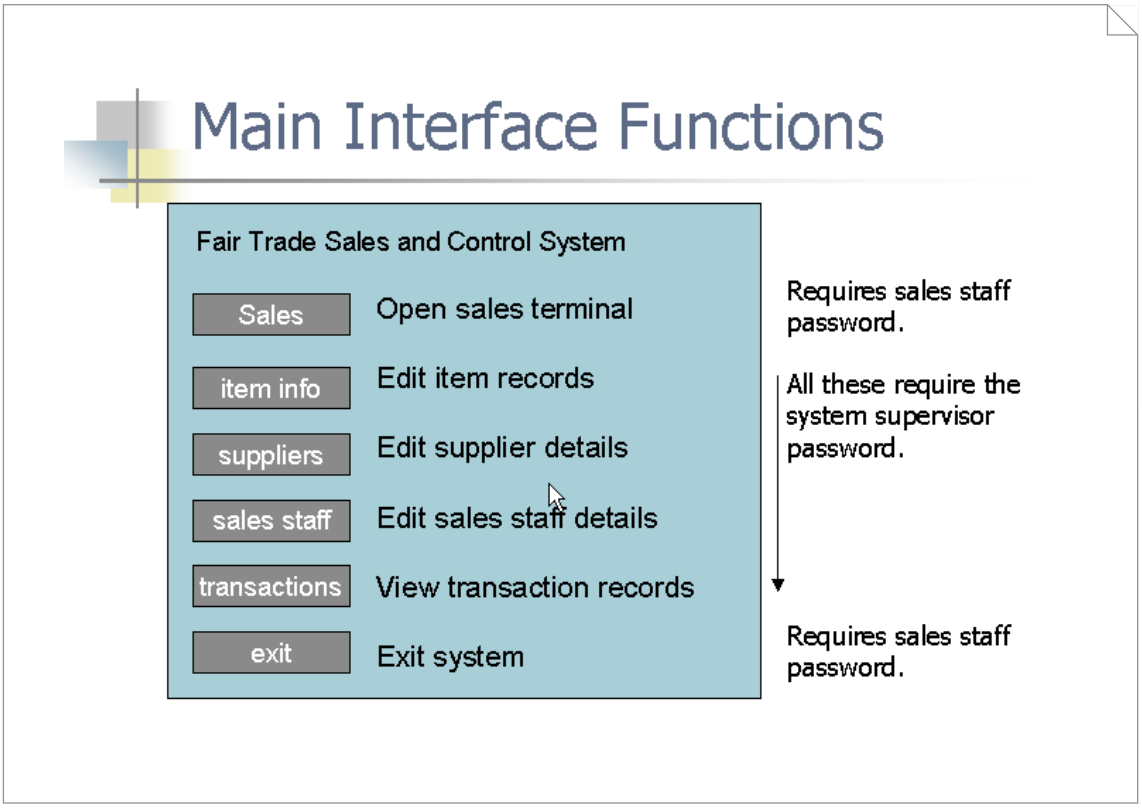
```
1.2 View all Flight Details

ID number: 234
First name: Nat
Last name: Gullayanon
Boarding house: Kamala
Year: 13
Flight number: TG 345
Origin: Phuket
Destination: Bangkok
Departure date: 23/7/2004
Departure time: 12:00 AM
Arrival time: 12:00 PM

          Press Enter to continue...
```

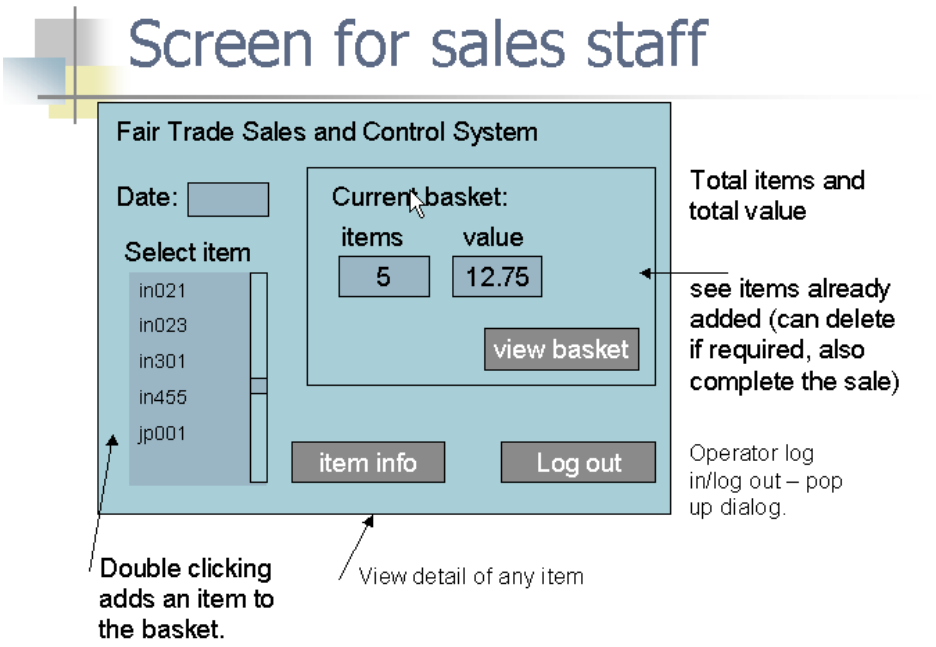
Diagram 6: This screen will show all the flight details of the entire student in every Boarding House as mentioned before. The record will be sorted in ID number ascending order. (P1.2)

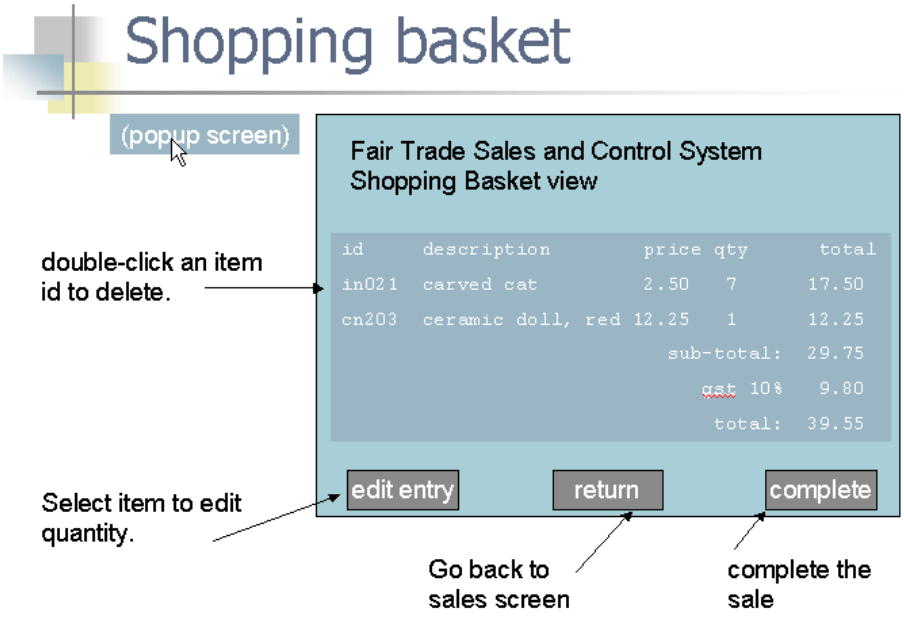
The following is an example of a sketched prototype.



The following prototypes were produced using PowerPoint.



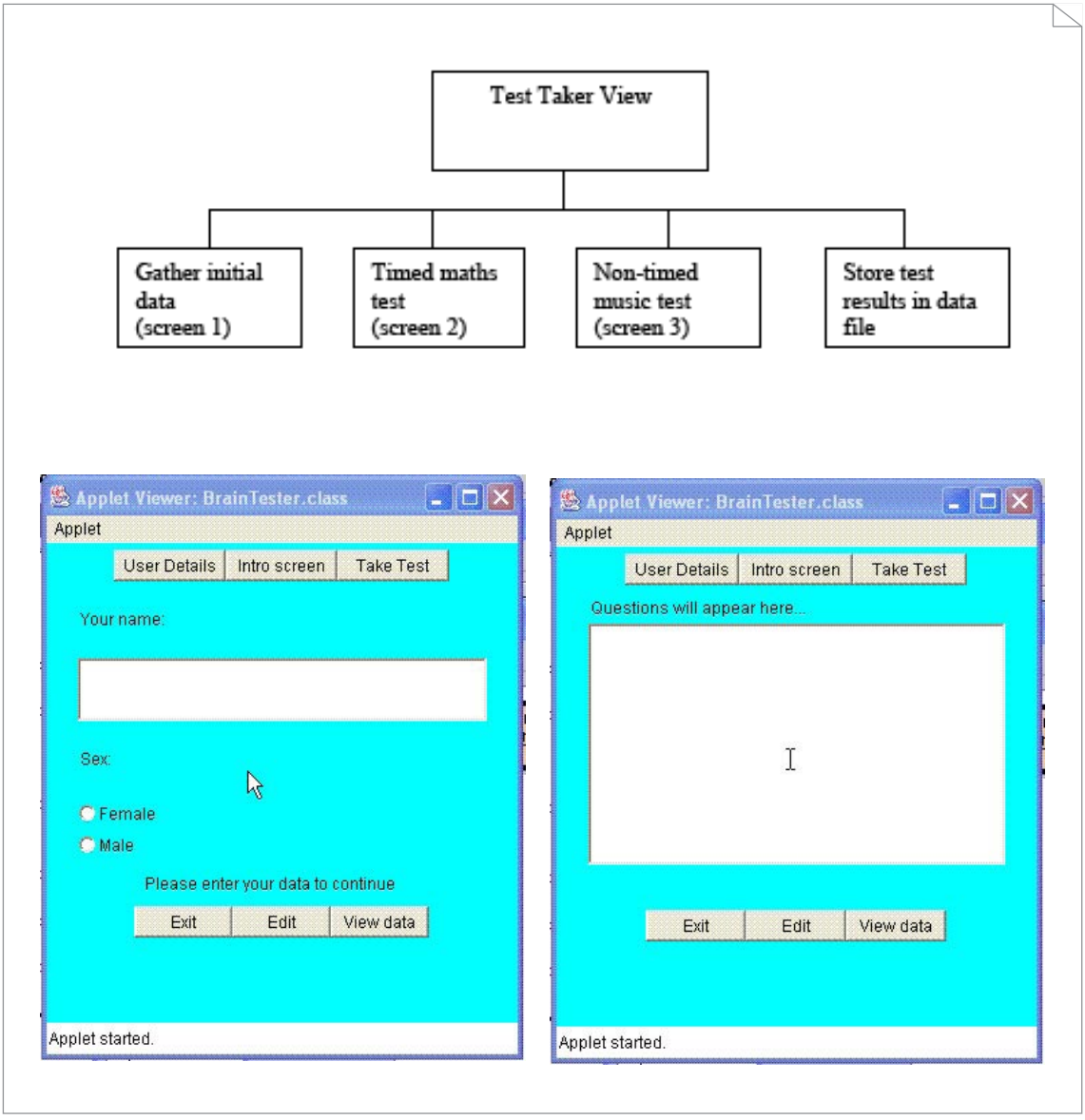




Level 2

Achievement level	Descriptor
2	The student includes an initial design and a prototype that **corresponds**.

This section shows an initial design and part of the prototype that was produced as a non-functional GUI applet.



One advantage of the applet prototype is that it gives additional insight into a Java-based solution.

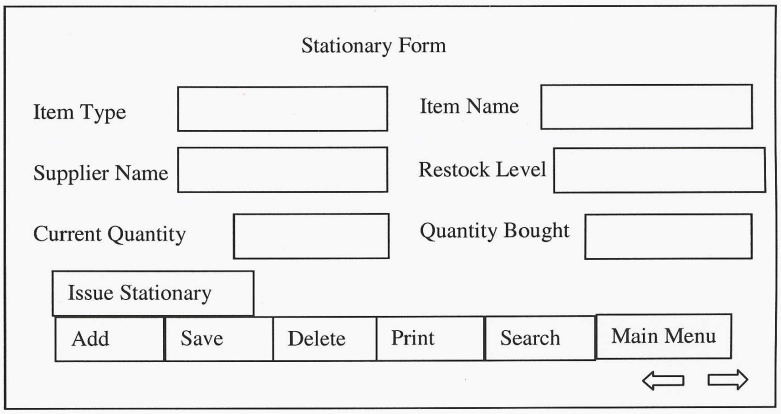
Level 3

Achievement level	Descriptor
3	The student includes an initial design and a complete prototype that corresponds to it **and documents user feedback** in evaluating the prototype.

There are also a large number of ways in which user comments can be documented. The easiest might be user comments written directly onto the prototype screens.

Projected Stationary Form

This form allows the user to perform standard processes (add, delete, save, print, search) The item type will be a choice between books (writing pads, jotter, notebooks etc, and stationary like pens paper etc)



User Comments

What is "Issue Stationery" - a button
Suppose it is a box of pencils or just one? How to tell?
How does the search work? Can I type search data into any box??
Quantity Bought doesn't seem to belong on this form?

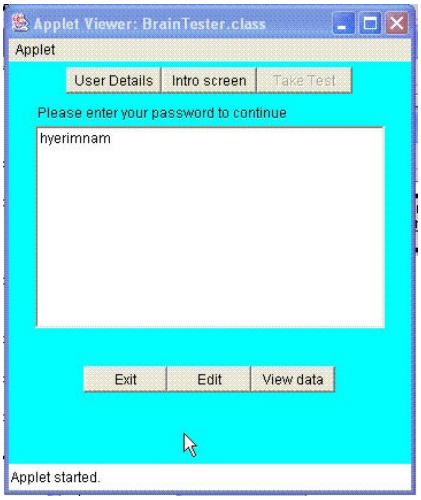
21

This next example reports the results of an interview. This led to a revision of the initial goals and helped to clarify the requirements, as intended.

== Discussion with User ==
The user asked the following questions:
· *How complicated can the formulas be? What about integrals and surds?*
- Response: The programmer doesn't know how to do this, so will stick to simpler formulas.
Perhaps some complicated symbols/formulas will be added, but no promises.
Which software tools can I use? Will I be able to add my own software tools?
- Response: Yes, we can make it possible to add extra tools.
· *If this program makes web-pages, will I need to learn about Front Page to put them on the W*eb?
- Response: You will need to do the site management yourself (e.g. uploading, making links,etc)
· *What about old IB exams and textbooks? Can I use a scanner?*
- Response: Yes, the scanner software will just be another tool in the expandable list.
· *How will the problem database be organized?*
- Response: This led to a long discussion. The set of topics is different for different courses, limiting this to specific categories seems too rigid. The agreement was to allow a set of keywords" to be added to each problem, then do searches on these keywords.

== Revised Goals ==
· Formulas are easy to type
- but this will be limited to fairly simple formulas using polynomials and fractions
· Some complex formulas can be typed
- only if the programmer manages to figure out how to do this.
· Special math symbols are included
- using names like, e.g. `pi, `e, etc.
· Resulting output is correct, readable, and properly formatted
- as HTML source code
· Various tools are better integrated
- extra tools can be added by the user
· Combining pictures and text is easy
- next to each other, using tables
· Numbers (or other information) in problems can be easily changed
- by changing the scripts
· Automated searching for problems matching a topic
- by adding keywords to each problem
· Assemble tests from selected problems
- the resulting test consists of scripts
· Store tests for further editting
· Produce properly formatted web-page versions of tests
- as standard HTML - no web-site management, except with using existing tools.

It may be possible to present prototype screens and comments together.

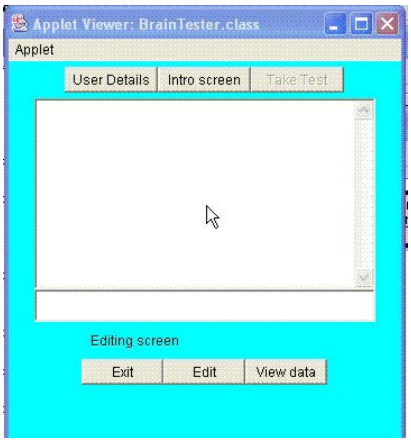


The password entry screen. Probably a traditional password dialog box will be used in the final implementation.

The Edit and View data buttons don't function if the password is not entered.

User Comments

What is the password for? OK, we agreed that there is no need for a very secure password (although I will if I can).



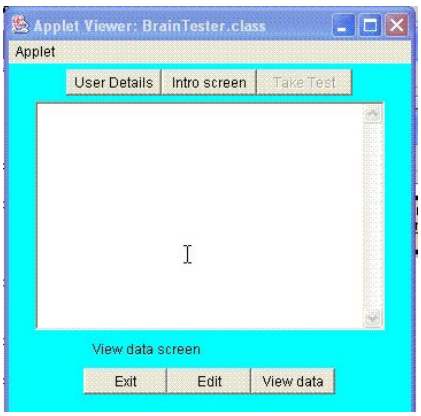
Basic editing screen. Intro text will be in the top window; questions in the bottom.

Probably the TakeTest button will double up as a "next" button again to cycle through the questions for editing.

User Comments

How do the questions appear; where is the space for the answer? Good point! Oops; needs more thought on this screen design.

Need buttons to cycle through questions, a save option and an option to change the music file – this is where a config file would be useful. Might have to re-think this design.



Basic view data screen. The user will specify the format later.

User Comments

We agreed a columnar look for the data:

name	sex	score	answers to music q's	time
fred	m	21	0011001111	23
jane	f	30	1100000101	122

and so on. Good idea about the bit representation of the true/false answers!

Stage B—Detailed design

The ordering of the criteria B1–B3 does not imply that this is the sequence in which students should develop or document their designs. This will vary according to the methodology adopted.

Criterion B1: Data structures

Students should choose data structures, at the design stage, that fully support the data-storage requirements of the problem, and that allow clear, efficient algorithms to be written. The data structures must fully support the objectives of the solution (criterion A2). The classes chosen should be logical in that the data is sensible for the objects in question and the methods are appropriate for the data given. This section of the program dossier could include class definitions, file structures, abstract data types (particularly at higher level) and some consideration of alternatives.

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the data structures/types to be used in the solution.
2	The student has **described some** of the data structures/types to be used, and provided sample data.
3	The student has discussed and **clearly illustrated all** of the data structures/types to be used to solve the problem, and provided sample data for **all** of them.

This section would typically be two to five pages in length.

Data structures and data members that are to be used in the programmed solution should be discussed here. Sample data, sketches/illustrations, including discussion of the way data objects will be changed during program execution should be demonstrated to achieve a level 4 in criterion B1.

Level 0

Achievement level	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.

The following is an example of a level 0 data structures section. The example indicates a minimal attempt to discuss the data structures used in solving the problem that has been identified. Much of the discussion here is concerned with a function that opens a file rather than the internal structure of the file itself or even what might be stored in it.

In this program, double linked lists are used. This is efficient choice for keeping quite a big data because the data could be implemented easily using the linked list. The pointer in the linked list would easily transfer the data stored in the linked list. The use of link would also allow easy comparison of the two data since the new link could be declared easily. Using of structures allow easy storage of the data.

Also, use of the function open file is very appropriate data structure because through the function, the list of passengers could be called. However, without the function, the data used in the program must be shown in the coding. It would be very inefficient to write big list of groups directly to the code.

Example:

```
thelist.openfile() //load file to masterlist

thelist.display();
```

Calling of the function increased the efficiency of the program since without it, the data used in the program had to be pre-declared in the code.

Level 1

Achievement level	Descriptor
1	The student has **outlined some** of the data structures/types to be used in the solution.

This next example is good, but lacks sample data and a thorough description.

Arrays

The user's original requirement was for 30 maths questions and 10 questions on the music. Clearly an array of questions is easier to process in the program than a series of individual Strings.

It would be nice if the answers could be worked out automatically but, this is quite complex since the numbers and operators would need to be stored separately. This is why a String array has been used.

The answers will be stored in a parallel array of ints:

Data Files

I shall use simple text files for i/o. I didn't use them in the prototype but there does not seem to be any problem. I had constructed a file using Notepad like this:

Each piece of data will be written into a text file and separated with a comma, this file can then be exported and used as a csv file in Excel:

```
"name", "sex", "score", "music"
 "Richard ", "m", 29, 100111011011
"Chanika ", "f", 32, 101110101001
```

```
Each time the test is taken, another line will be added to this file.
The bit representation can be held in an array of characters:

The only other file of interest is the wav file used to store the
music.  The user wasn't too happy about my suggestion of copying a
file called "music.wav" into the application directory so I will need
a method of storing a wav filename for the music.
```

The inclusion of the data to be written to the CSV file is insufficient for a level 2 award since it is only a single example (no data is shown for the question arrays mentioned earlier).

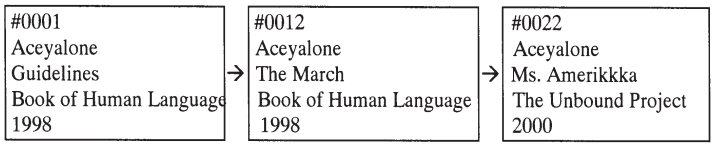
Level 2

Achievement level	Descriptor
2	The student has **described some** of the data structures/types to be used, and provided sample data.

The following (partial) example illustrates a description that includes sample data.

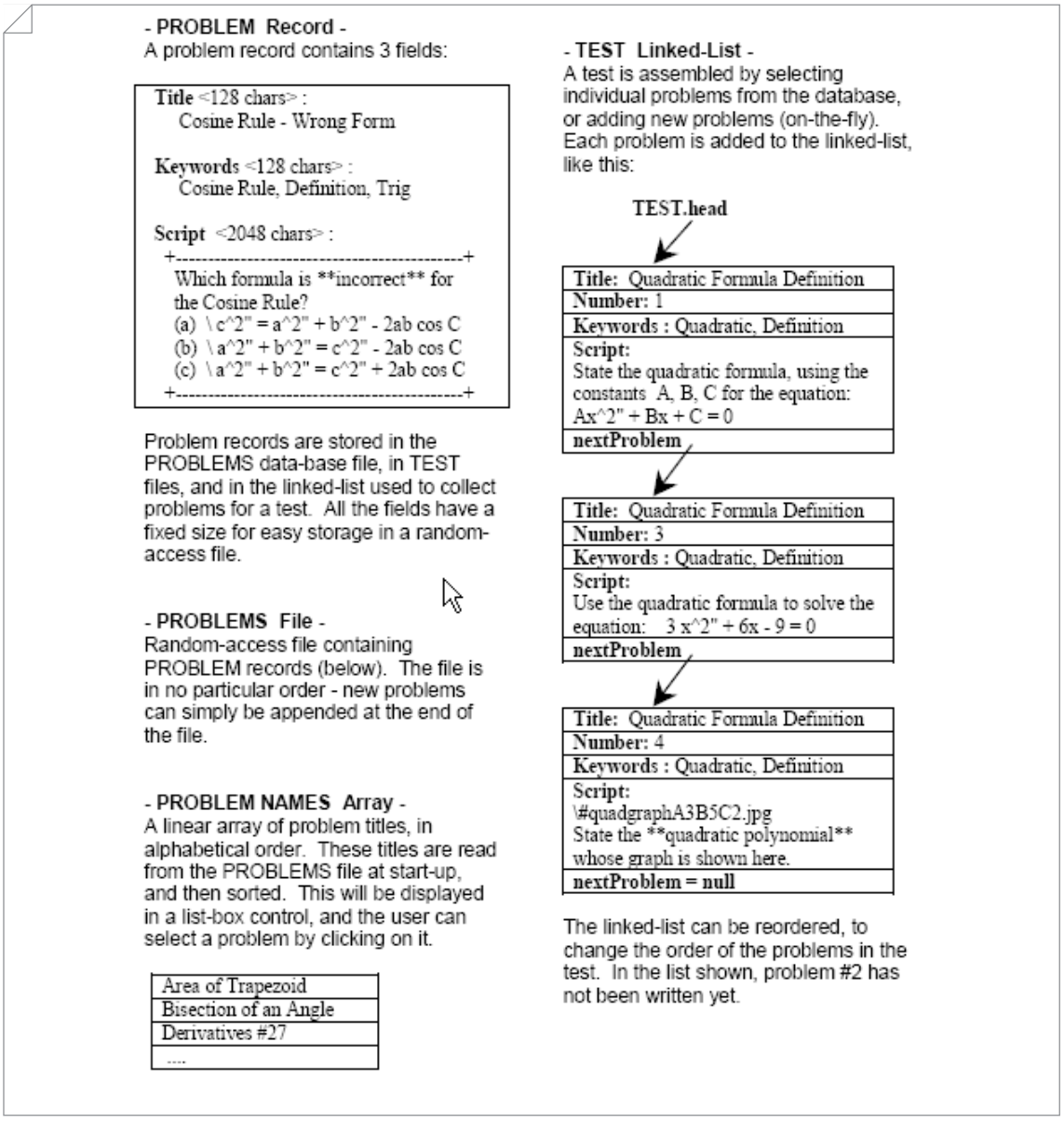
```
DATA STRUCTURES

For my dj helper, I have created a class called Track which consists
of artist name, song title, album name, year of release, an optional
comment for each song, a set of criteria which will be set by the
user to help them in their search, and finally the Class will include
a reference to the next track. Each track will be given an ID# which
will allow it to be found easily later on. This data structure will
be used in my search database function, each track will be created in
create track and each track can be edited or deleted. The tracks can
also be stored to playlists where each playlist is just a collection
of ID#'s.

The file "catalog" will store all of the tracks entered by the user
in the order they put them in. Playlists are linked lists that are
created by grabbing the required tracks from the file "catalog" based
on their ID#'s.

Class Track data members:
{
  private String artist;
  private String title;
  private String album;
  private int year;
  private int[5] crit;
  private Track root;

  methods..

}
```

Here is an example of a playlist containing only tracks from Freestyle Fellowship member and Los Angeles underground legend Aceyalone:



The name of this list, "Acey", will be saved to a file where the ID#'s will be stored and then read from the "catalog" file when this playlist is called by the program. At this point, the artist name, song title, album name, and year will be read out to the user.

In the next example, the student presents sample data and sufficient detail about the data structures used. However, some structures remain to be illustrated and discussed.



Level 3

Achievement level	Descriptor
3	The student has discussed and **clearly illustrated all** of the data structures/types to be used to solve the problem, and provided sample data for **all** of them.

For an award of level 3 the student should provide the following:

- diagrams of all the structures used in the program (arrays, records, files). At HL, this includes any abstract data types (ADTs) that have been used
- a discussion of why particular data types and structures are suitable to the problem at hand
- a mention of some other data structures that could have been used
- illustrations of the way the storage structures change over time as data is added and removed
- at SL, a discussion/explanation of the simple data types chosen.

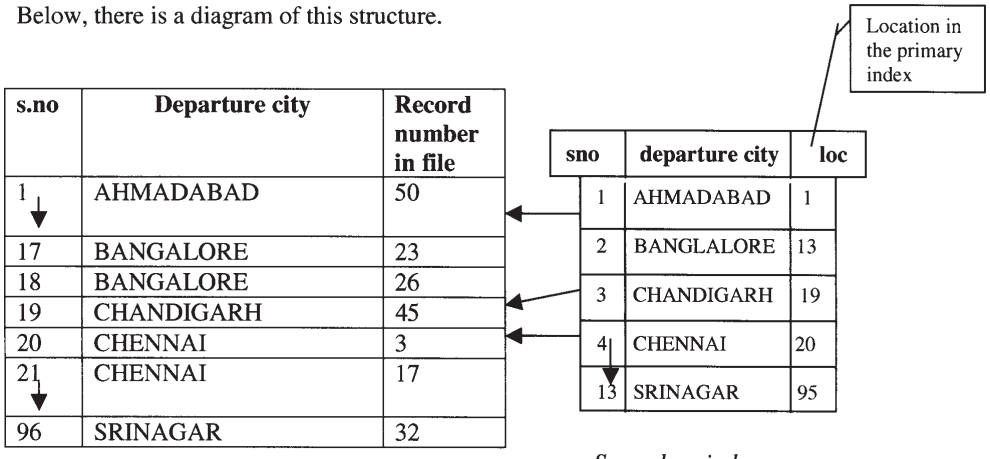
STORING THE FLIGHT DETAILS

Initially I thought of using a **partially indexed file with two levels of indexing** to store the fight details of all the flights.

In this kind of a structure, the records are first stored in ascending order of flight numbers in the file. Then an index is created (primary index) which stores all the flights in alphabetical order of departure cities. *So, for example, the first element of this primary index, as in the diagram below, has its departure city as Ahmadabad but it is actually the 50th record in the file.* This way, all the flights are stored.

After this is created, a secondary index is created which stores all the departure cities only once and has a field, "bc", which stores the location of the first occurrence of that departure city in the primary index.

Below, there is a diagram of this structure.



Primary index

Secondary index

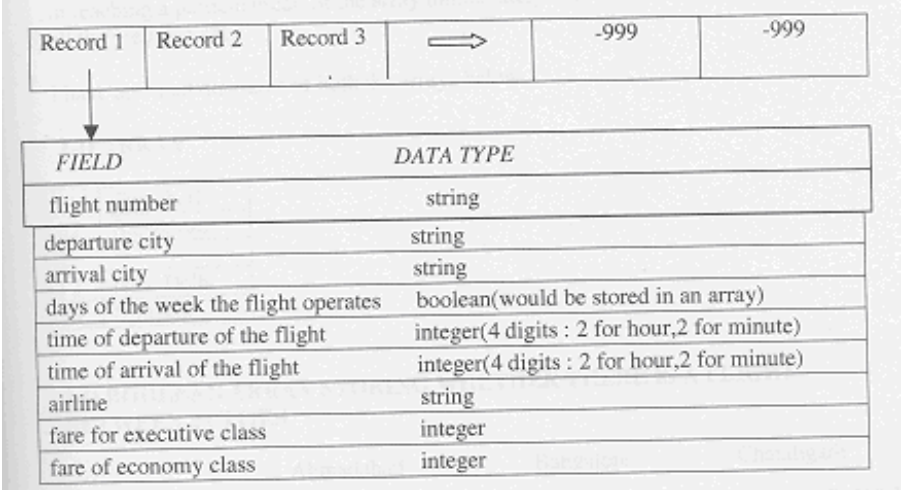
This solution is a good one but is very complex and is hard to implement. The efficiency of searching for a flight using this solution would be $O(n_i + n_j)$ where 'i' is the block size of the secondary index and 'j' is the block size of the primary index. Since I am implementing a linear search to locate the city wanted in the secondary index, the efficiency would depend on the number of elements searched before finding the right city. And once the city is located, the efficiency would depend on the size of the block to be searched to locate the required flight.

This is quite efficient as i and j would not be too large. But to add a new record to this system would include adding it to the file and then updating both the indexes which is a laborious process. That is why I started facing problems with this solution; so I decided to look for a simpler solution.

I figured that the next best way to create a database of flights and their details was to store each flight as a record in a sequential file. The file could be maintained in ascending order of flight number. An "end of file marker " of -999 could be assigned. A *binary search* could be implemented to locate any record using the flight number. To *add a record*, the insertion point could be found using a linear search and then all the records could be shuffled to make space for the new record.

To *delete a record*, the binary search could first search for the record to be deleted, and then all the record could be shuffled to cover up the empty space created by the deleted record. The last record which is now empty is given the value —999.

I have attached a picture of the file structure on the next page.



Record 1	Record 2	Record 3	⟹	-999	-999

FIELD	*DATA TYPE*
flight number	string
departure city	string
arrival city	string
days of the week the flight operates	boolean(would be stored in an array)
time of departure of the flight	integer(4 digits : 2 for hour,2 for minute)
time of arrival of the flight	integer(4 digits : 2 for hour,2 for minute)
airline	string
fare for executive class	integer
fare of economy class	integer

Each record in the file contains all these fields which are of the types which are mentioned next to them respectively. An example of a record in the file would be:

865 DELHI GUWAHATI thu/sat 010 1225 Indian Airlines 9990 6725

The records in the file are ordered according to their flight numbers. The diagram below illustrates this:

Record number	Flight number	Airline
1	104	Jet Airways(9W)
2	106	Indian Airlines(IC)
3	110	Indian Airlines(IC)
↓		
95	3401	Jet Airways(9W)
96	3518	Jet Airways(9W)

Thus, storing the records in a file is quite convenient and efficient. The efficiency of adding a record to the file would be 0(n) as it just involves shuffling. However, the file size is quite large. But this was still the most convenient solution to implement and it works quite well.

IS THERE A FLIGHT BETWEEN 'place a' AND 'place b'?

I have used a 1 D array and a 2D array to easily detect whether there is a flight from a departure to arrival city. The one dimensional array stores all the cities one after the other. The 2D array checks whether there is a link between 'place a' and 'place b'.

Even if there isn't a direct flight from 'place a' and 'place b', this 2D array helps to detect whether it is possible to reach the destination with a stop over at some other place. Also, the 1D array of cities makes the process of locating flights easy as it aids in reaching a particular cell of the array immediately without having to scan through all the previous ones.

I have attached diagrams of both the arrays below.

1-D ARRAY

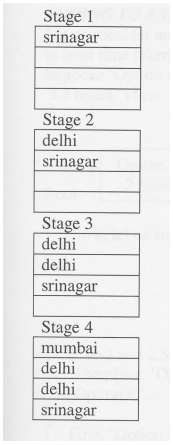
01	Guwahati
02	Bangalore
12	New Delhi
13	Srinagar

2-D BOOLEAN ARRAY STORING WHETHER THERE IS A FLIGHT BETWEEN 2 CITIES

◊	Ahmadabad	Bangalore	Chandigarh
ahmadabad	F	T	F
Bangalore	T	F	F
Chandigarh	F	F	F
Chennai	F	T	F
Mumbai	T	T	F
New Delhi	T	T	T
Srinagar	F	F	F

Lets consider an example now.
Suppose we had to go from New Delhi to Bangalore. We would first the position of both these cities in the 1 D array, which are 12 and 3 respectively. With this information, we can straightaway go to the highlighted cell in the 2D array and check whether the link is T or F. Since it is T, we know that there is a flight between New Delhi and Bangabore.

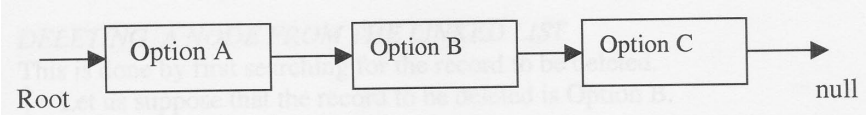
STACK STORING A SERIES OF DEPARTURE AND ARRIVAL CITIES



The stack is used in my program to store arrival and departure cities of fights whenever a suitable flight option is found using the 1 D and 2D arrays.
For example, if to go from Mumbai to Srinagar, a good flight option *Mumbai — Delhi — Srinagar* is discovered using the 2 D array, then Srinagar,Delhi,Delhi and Mumbai are pushed onto a stack respectively. So, when flights are being searched for, then 2 cities are popped at a time and a search to find a flight from the first city to the second is carried out till the stack is empty.

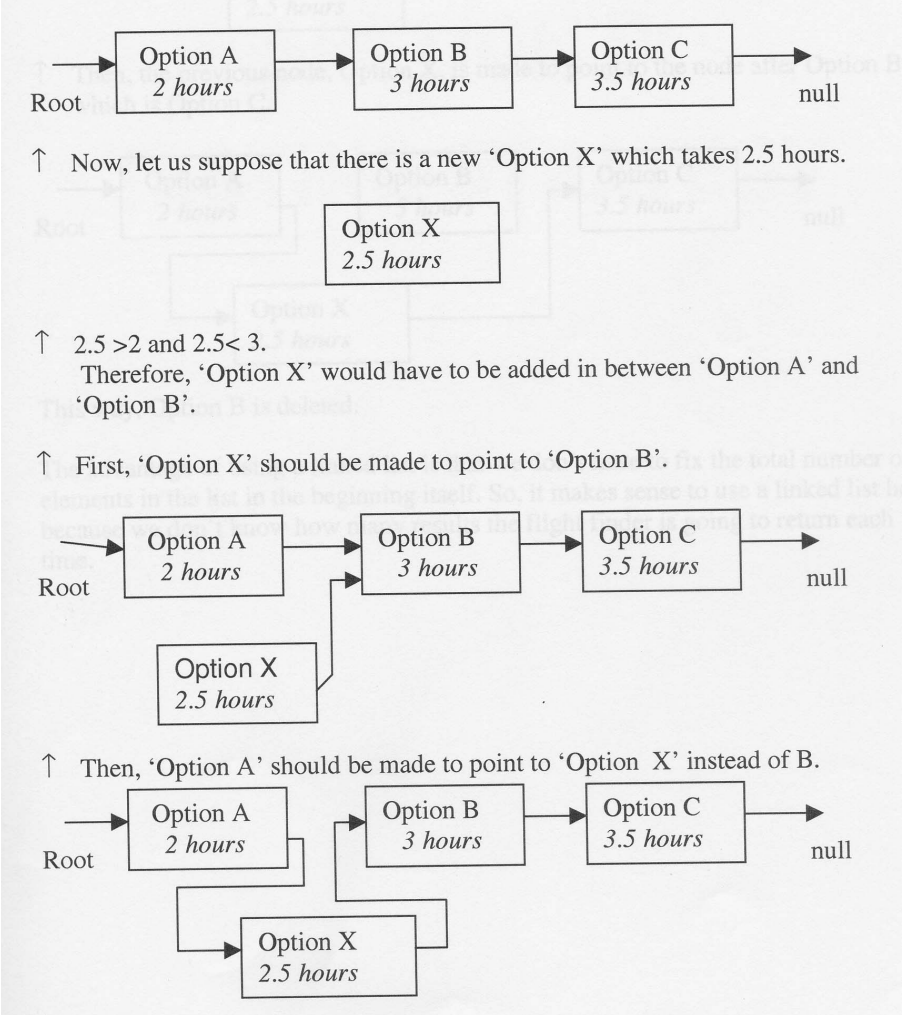
LINKED LIST TO DISPLAY THE FLIGHT OPTIONS

My final data structure would be a linked list which would be used to display the flight options eventually. However, before that, it stores the flight options in a sorted order, according to **least time** or **best price** options.



ADDING TO A LINKED LIST

Let us consider an example where the user wanted the options to be ordered according
to least time taken.
Suppose 'Option A' took 2 hours, 'Option B' took 3 hours and 'Option C' took
3.5 hours. Then:



So, that's how we add to the middle of a linked list. It is important to first point Option X to Option B and then only point Option A to Option X because we will otherwise lose the position of Option B.

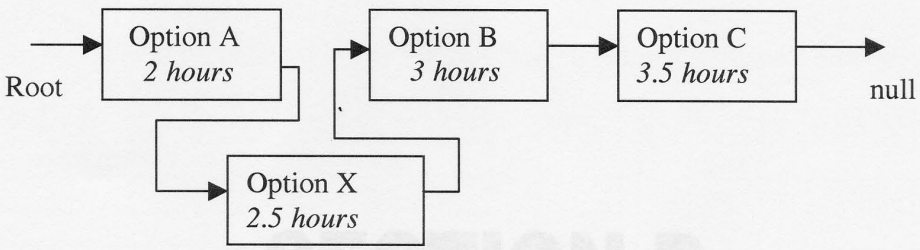
SEARCHING FOR A NODE IN THE LINKED LIST

This is done by traversing down the linked list right from the first record until the record with the wanted flight number is found.

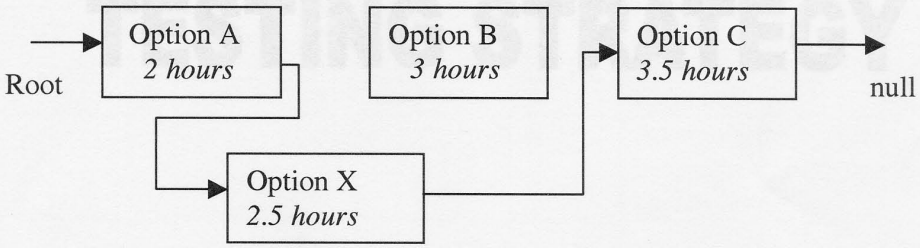
DELETING A NODE FROM THE LINKED LIST

This is done by first searching for the record to be deleted.

↑ Let us suppose that the record to be deleted is Option B.



↑ Then, the previous node, Option X, is made to point to the node after Option B which is Option C.



This way, Option B is deleted.

The advantage of using a linked list is that we don't have to fix the total number of elements in the list in the beginning itself. So, it makes sense to use a linked list here because we don't know how many results the flight finder is going to return each time.

Criterion B2: Algorithms

Students should choose algorithms, at the design stage, that fully support the processes needed to achieve the objectives of the solution (criterion A2), and provide sufficient support for the required data structures. The classes chosen should be logical in that the methods are appropriate for the data given. Students must include parameters and return values.

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the algorithms to be used in the solution.
2	The student has **described most** of the algorithms to be used, with details of parameters and return values.

This section would typically be two to five pages in length.

This can be a list or outline of all the algorithms, presented as text, possibly in outline format. Standard algorithms (such as search or sort) can simply be named (with parameters), but non-standard algorithms must be described in more detail.

Level 1

Achievement level	Descriptor
1	The student has **outlined some** of the algorithms to be used in the solution.

The following example demonstrates an outline of an algorithm.

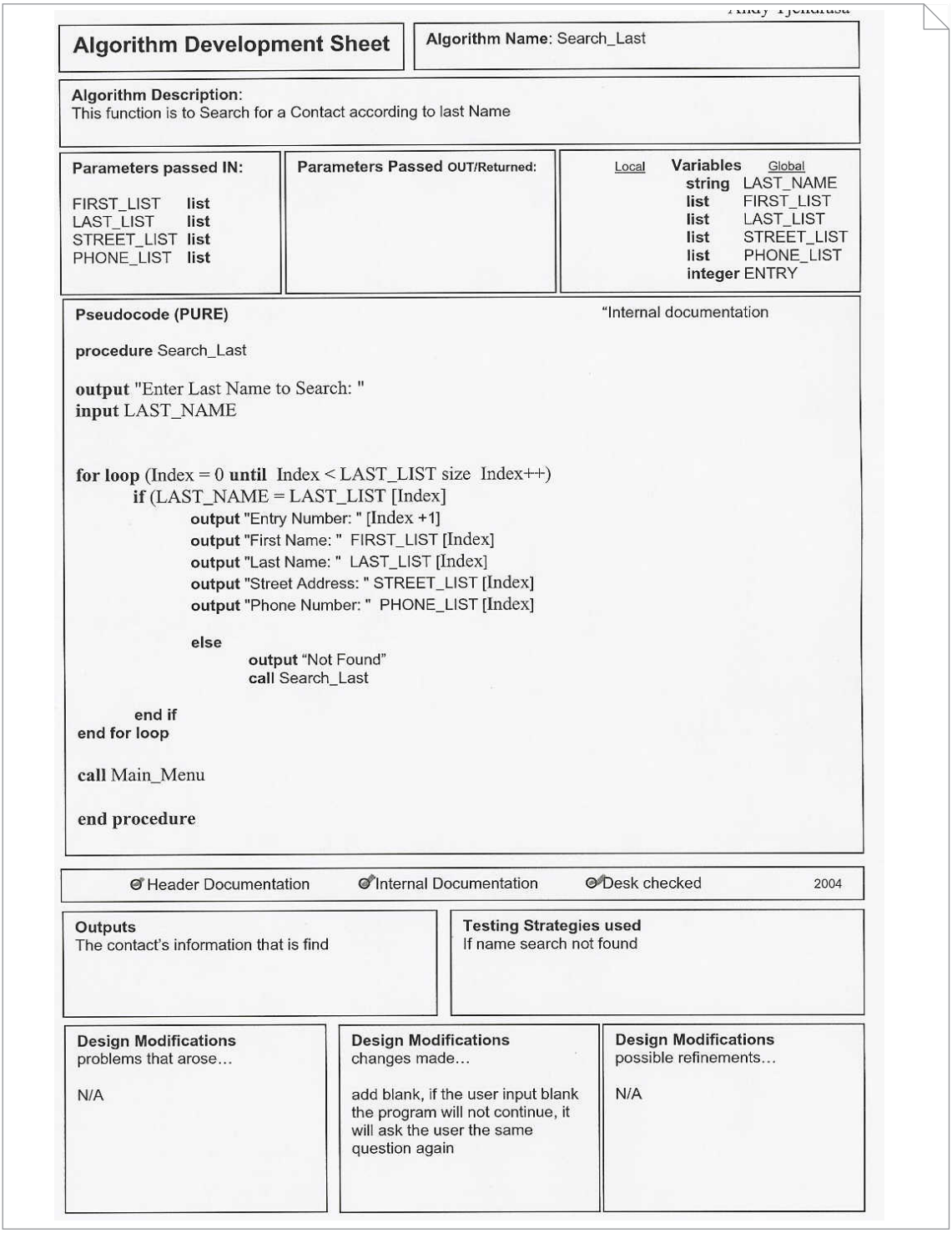
iii. Create linked list

```
function newnode (val c as integer) result integer
declare ptr, ptr1 as pointers
ptr←NIL
declare cfptr, cfptr1 as pointers
cfptr←NIL
if(c=1) then
if(ptr=NIL)then
        ptr←allocate(nodo)
        output("insert product")
        input(b)
        ptr→prod←b
        output("insert flavour")
        input(g)
        ptr→tipo←g
        output("insert container")
        input(h1)
        ptr→envase←h1
         output("insert size")
         input(i)
          ptr->tam<-i
        ptr→ptr←NIL
  endif
```

Level 2

Achievement level	Descriptor
2	The student has **described most** of the algorithms to be used, with details of parameters and return values.

The sheet shown below represents one approach to coding algorithms that is very thorough. This was produced for PURE but could easily be adapted for Java.



Algorithm Development Sheet

Algorithm Name: Search_Last

Algorithm Description:
This function is to Search for a Contact according to last Name

Parameters passed IN:

FIRST_LIST **list**
LAST_LIST **list**
STREET_LIST **list**
PHONE_LIST **list**

Parameters Passed OUT/Returned:

Local **Variables** Global
string LAST_NAME
list FIRST_LIST
list LAST_LIST
list STREET_LIST
list PHONE_LIST
integer ENTRY

Pseudocode (PURE) "Internal documentation

```
procedure Search_Last

output "Enter Last Name to Search: "
input LAST_NAME


for loop (Index = 0 until  Index < LAST_LIST size  Index++)
        if (LAST_NAME = LAST_LIST [Index]
                output "Entry Number: " [Index +1]
                output "First Name: "  FIRST_LIST [Index]
                output "Last Name: "  LAST_LIST [Index]
                output "Street Address: " STREET_LIST [Index]
                output "Phone Number: "  PHONE_LIST [Index]

                else
                        output "Not Found"
                        call Search_Last

        end if
end for loop

call Main_Menu

end procedure
```

Header Documentation Internal Documentation Desk checked 2004

Outputs
The contact's information that is find

Testing Strategies used
If name search not found

Design Modifications
problems that arose…

N/A

Design Modifications
changes made…

add blank, if the user input blank the program will not continue, it will ask the user the same question again

Design Modifications
possible refinements…

N/A

Criterion B3: Modular organization

Students should choose modules, at the design stage, that incorporate the data structures and methods required for the solution (criteria B1 and B2) in a logical way. The data structures must fully support the objectives of the solution (criterion A2). Students must present this organization in a structured way that clearly shows connections between modules (hierarchical decomposition or class dependencies). The connections between modules, algorithms and data structures must also be presented.

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the modules to be used in the solution.
2	The student has **described most** of the modules to be used, **showing connections** between them.

This section would typically be three to five pages in length.

A variety of presentations are possible here. Some possibilities are:

- a top-down hierarchical decomposition chart containing the names of modules, showing connections between modules and showing details of which data structures and methods are connected with (or part of) which modules
- a text outline showing hierarchical decomposition (equivalent to above)
- a hard copy of CRC cards showing dependencies between collaborating classes, with details of which data structures and methods are connected with (or part of) which classes.

The design is assessed independently from the programming stage (stage C). The design should be complete, logical and usable, but the student may deviate from it or expand it during stage C, without penalty.

Level 0

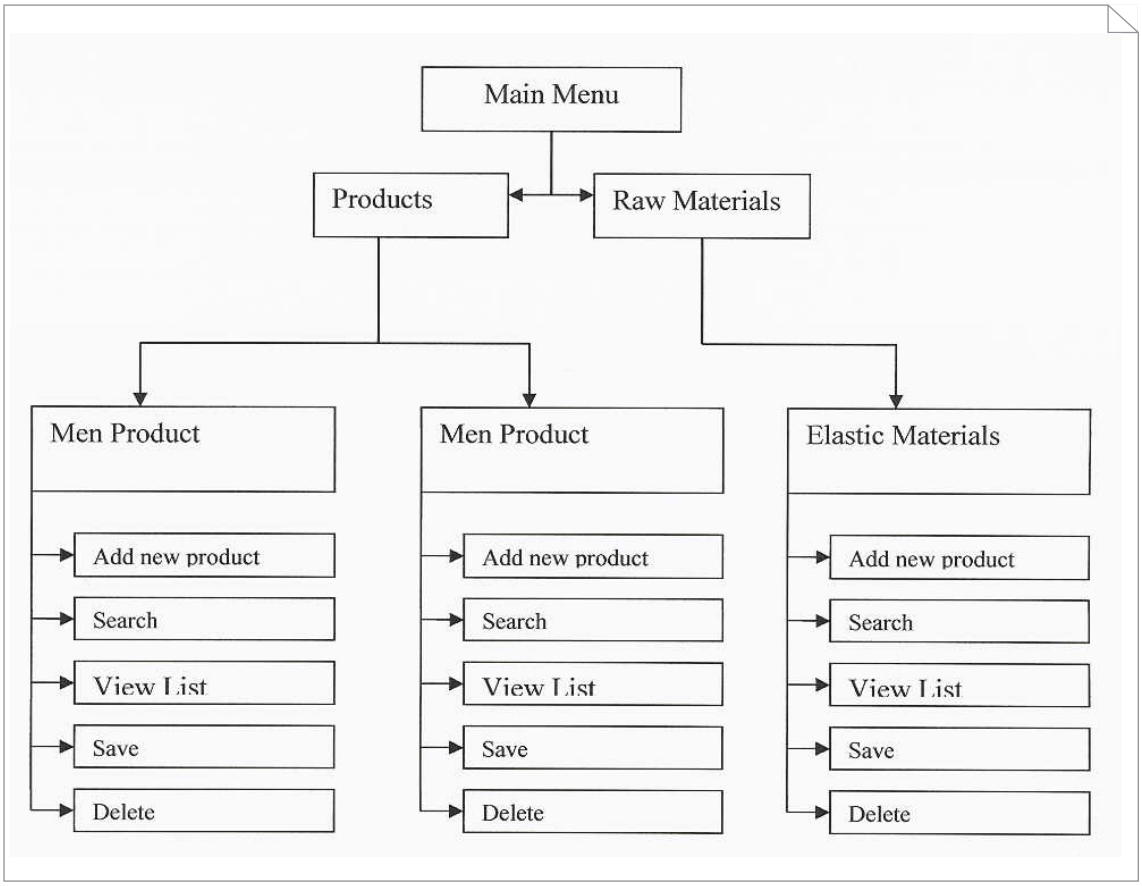
Achievement level	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.

Since the level 1 criterion refers to "some of the modules", students could be awarded this level even if one or two modules are presented without any connections being shown.

Level 1

Achievement level	Descriptor
1	The student has **outlined some** of the modules to be used in the solution.

This example shows two main modules, which are further subdivided. The details are clearly not well thought out, but will be acceptable as an outline.



Level 2

Achievement level	Descriptor
2	The student has **described most** of the modules to be used, **showing connections** between them.

The following example illustrates a text description of some modules.

View Flight Details module:

In this module contains four menu choices where user can choose how they are going to view the flight details and a return choice. The choices that can be called are Quick search Flight Details, View all Flight Details module, View each Boarding House Flight Details module and return to Main Menu module.

Called from	Links to
1. Main module	1. Quick Search Flight Details 2. View all Flight Details module 3. View specific House Flight Details module 4. Main module

Quick Search Flight Details module:
(Involved passing head pointer parameter as a data structure)

This module will read in user input ID number that they wish to search for flight details. This module will then search for the record in the data structure with identical ID number using pointer. If the record is found, it will then display the flight details otherwise it will display error.

Called from	Links to
1. View Flight Details module	-

View all Flight Details module:
(Involved passing head pointer parameter as a data structure)

This module will display all the student flight record kept in the data structure in sorted order of ID ascending order.

Called from	Links to
1. View Flight Details module	-

View specific House Flight Details module:
(Involved passing head pointer parameter as a data structure)

In this module it required the users to pass in the Boarding House number they wish to view the Flight Details e.g. if the user select Rawai House then the this module will then read all the Rawai student flight records from the data structure and display them or otherwise display error if no record found.

The student also describes how modules connect to each other.

Stage C—The program

Program listings must contain **all** the code written by students and, if a program listing displays code that was automatically generated by the development system or copied from another source, then this code must be clearly identified and distinguishable from that code written by the students. Only the code **designed and written** by students must be taken into account when applying the assessment criteria.

Criterion C1: Using good programming style

Good programming style can be demonstrated by program listings that are easily readable, even by a programmer who has never used the program. These would include small and clearly structured Java methods, sufficient and appropriate comments, meaningful identifier names and a consistent indentation scheme.

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The program listing demonstrates **some** attention to good programming style.
2	The program listing **mostly** demonstrates attention to good programming style.
3	All parts of the program listing demonstrate **considerable** attention to good programming style.

A typical program should be approximately 1,000–3,000 (HL) or 500–2,000 (SL) lines of code in length.

Comments should be included to describe the purpose and parameters of each method, and also when code is difficult to understand.

The program should demonstrate the use of good programming techniques. It should include:

- an identification header indicating the program name
- author, date, school
- computer used, IDE used, purpose.

The program should possess good internal documentation, including:

- constant, type and variable declarations that should have explanatory comments
- identifiers with meaningful names
- objects that are clearly separated and have comments for their parameters
- suitable indentation that illustrates various programming constructs.

Generally, achievement level 2 will be appropriate where two or more of these have been demonstrated. Then, achievement level 3 will be appropriate for three or more being demonstrated.

Level 0

Achievement level	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.

It is increasingly hard to find code that does not pay at least minimal attention to good style.

Level 1

Achievement level	Descriptor
1	The program listing demonstrates **some** attention to good programming style.

```
  /**
   *TextField panel
   **/

  JPanel screenvarPanel = new JPanel();
  screenvarPanel.setLayout( new GridLayout( 10, 1 ) );
  id = new JTextField( 20);
  first = new JTextField( 20 );
  screenvarPanel.add( first );
  last = new JTextField( 20 );
  screenvarPanel.add( last );
  home = new JTextField("Enter number-click Find", 20);
  screenvarPanel.add( home );
  address = new JTextField( 20 );
  screenvarPanel.add( address );
  city = new JTextField( 20 );
  screenvarPanel.add( city );
  state = new JTextField( 20 );
  screenvarPanel.add( state  );
  zip = new JTextField( 20 );
  screenvarPanel.add( zip );
  country = new JTextField( 20 );
  screenvarPanel.add( country );
  email = new JTextField( 20 );
  screenvarPanel.add( email );
  fax = new JTextField( 20 );
  screenvarPanel.add( fax );

 /**
  *Accessibility Section
  **/

 lfirst.setLabelFor( first );
 llast.setLabelFor( last );
 lhome.setLabelFor( home );
 laddress.setLabelFor( address );
 lcity.setLabelFor( city );
 lstate.setLabelFor( state );
 lzip.setLabelFor( zip );
 lcountry.setLabelFor( country );
 lemail.setLabelFor( email );
 lfax.setLabelFor( fax );
 setLayout( new GridLayout( 1, 2 ) );
 add( labelPanel );
 add( screenvarPanel );
 }
}
```

In the previous example, the student has not used meaningful comments and the indentation is inconsistent. However, the identifiers' names seem to be reasonably well-chosen, and the main objects in this section have at least been separated with comments.

Level 2

Achievement level	Descriptor
2	The program listing **mostly** demonstrates attention to good programming style.

The following code is C++ but does at least illustrate excellent style.

```
        runner++;}
    gTimes.resize(runner);                  //resizes array to fit # of elements
}
//---------------------------------------------------------------------
void readInBNames(apvector<apstring> &bNames)
// pre: file must exist, in same folder, with proper name
// post: data in pre-existing file is read into program, so that information may be manipulated
{
    ifstream infile("boysNames.txt");       //reads in file
    if(!infile){
        cout << "File could not be opened." << endl;
        exit(1);}
    apstring temp;                          //used for name being read in from file
    int runner=0;                           //count used to increment position in array
    while(getline(infile, temp)){           //gets data from file
        bNames[runner]=temp;                //stores in array
        runner++;}
    bNames.resize(runner);                  //resizes array to fit # of elements
}
//---------------------------------------------------------------------
void readInBEvents(apvector<apstring> &bEvents)
// pre: file must exist, in same folder, with proper name
// post: data in pre-existing file is read into program, so that information may be manipulated
{
    ifstream infile("boysEvents.txt");      //reads in file
    if(!infile){
        cout << "File could not be opened." << endl;
        exit(1);}
    apstring temp;                          //used to store event name from file
    int runner=0;                           //count used to increment array
    while(getline(infile, temp)){           //reads in line from file
        bEvents[runner]=temp;               //stores in array
        runner++;}
    bEvents.resize(runner);                 //resizes array to fit # of elements
}
//---------------------------------------------------------------------
void readInBTimes(apvector<int> &btimes)
// pre: file must exist, in same folder, with proper name
// post: data in pre-existing file is read into program, so that information may be manipulated
{
    ifstream infile("boysTimes.txt");       //reads in file
    if(!infile){
        cout << "File could not be opened." << endl;
        exit(1);}
    int runner=0, time;                     //count used to increment array
    while(infile >> time){                  //reads in line from file
        btimes[runner]=time;                //stores in array
        runner++;}
    btimes.resize(runner);                  //resizes array to fit # of elements
}
//---------------------------------------------------------------------
void gResize(apvector<apstring> &gNames, apvector<apstring> &gEvents,
          apvector<int> &gTimes, int &function)
// pre:  arrays must exist and are previously defined
// post: girls arrays are resized properly, according to what function gResize was called from
{
    int length=gNames.length();
    if(function==2){
        gNames.resize(length+1);
        gEvents.resize(length+1);
        gTimes.resize(length+1);}
    else if(function==6){
        gNames.resize(length-1);
        gEvents.resize(length-1);
        gTimes.resize(length-1);}
}
//---------------------------------------------------------------------
void bResize(apvector<apstring> &bNames, apvector<apstring> &bEvents,
          apvector<int> &bTimes, int &function)
// pre: arrays must exist and are previously defined
// post: boys arrays are resized properly, according to what function bResize was called from
{
    int length=bNames.length();
```


Level 3

Achievement level	Descriptor
3	All parts of the program listing demonstrate **considerable** attention to good programming style.

The award of a level 3 would depend upon the thoroughness with which all parts of the listing are presented in a style similar to the level 2 example given above. The criteria are quite specific about what is needed to distinguish between these two levels.

The program should possess good internal documentation, including:

- constant, type and variable declarations that should have explanatory comments
- identifiers with meaningful names
- objects that are clearly separated and have comments for their parameters
- suitable indentation that illustrates various programming constructs.

Generally, achievement level 2 will be appropriate where two or more of these have been demonstrated. Then, achievement level 3 will be appropriate for three or more being demonstrated.

Criterion C2: Handling errors

This refers to detecting and rejecting erroneous data input from the user, and preventing common run-time errors caused by calculations and data-file errors. Students are not expected to detect or correct intermittent or fatal hardware errors such as paper-out signals from the printer or damaged disk drives, or to prevent data-loss during a power outage.

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student **includes** documentation that shows a **few** error-handling facilities in the program, or documents only **one** type of input or output.
2	The student includes documentation that shows **many** error-handling facilities in the program, and documents **more than one** type of input or output.
3	The student **fully** documents the error-handling of **each** input and output method within the program.

This section would typically be one to two pages in length.

For this criterion, students must attempt to trap as many errors as possible. The documentation in the dossier can take a variety of forms.

For example, students could highlight relevant comments within the program listing or they could produce a table with two columns, one that identifies any error possibilities, and one that shows the steps taken to trap the errors. It is not expected that extra output is produced for this section.

Level 0

Achievement level	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.

This level will be awarded to dossiers where no section on error handling is present at all. If the student chooses to use a highlighter on the program listing with annotations as comments, which can be made by hand, then this section could consist simply of a list or table referencing those comments.

Level 1

Achievement level	Descriptor
1	The student **includes** documentation that shows a **few** error-handling facilities in the program, or documents only **one** type of input or output.

This next example was presented as a complete section on error handling (note the code is not Java).

Error Handling

The program is not the only one who can make mistakes, the user can do so as well. So when programming the programmer should be aware of this and help the user to not commit errors. User friendliness is used here as well but this will not prevent the program from crashing when an invalid input is entered. When this happens the program should have some kind of protection in order not to crash.

I have implemented the following procedure in my menu so that when an option different than the ones given is entered the program will not crash.

```
procedure intreadln(var intvar:integer); var stringvar: string;
code:integer;
begin

  repeat
    readln(stringvar);
    val (stringvar, intvar, code);
    if code<>0 then
      writeln('Enter option again');
  until code=0;
    end;
```

Since this student has produced a generalized routine that can be used in different parts of the same program, it is just possible to award a level 1 for this criterion. However, more detail is preferred along with actual examples of its use.

Level 2

Achievement level	Descriptor
2	The student includes documentation that shows **many** error-handling facilities in the program, and documents **more than one** type of input or output.

In the next example, the student documents (by their own admission) only some of the error-handling features of the program.

ERROR HANDLING FACILITIES:

Here are examples of the error handling facilities used in the program:

```
  Do{
  coutK<"\n\nDo you want to save changes before you exit (Y/N): ";
  cin>>Command;
  } while((Command ='Y')&&(Command ='y')&&(Command ='n')&&
          (Command ='N'));
```

Here, the user should only enter a "Y" for yes or an "N" for no. If something else was entered, an error will be generated if this do-while loop was not used.

```
  if(Choice==50) {
    if(X.length()>O)
    List(X,FlaggedEntries);
  else
    cout«"\nTHERE IS NO DATA STORED.\a\n\n";
```

Here, the program checks for entries before listing them. If there aren't any entries or records stored, the user will be told that there is no data stored. This will prevent errors from generating because if this wasn't done, the program will try to list some entries that do not exist and will generate an error.

```
  if(Choice==50) {
    if(X.lengthO>O)
      DeleteEntry(X);
    else
      cout«"\nTHERE IS NO DATA TO DELETE.\a\n\n";
```

When the user want to delete an entry, the program checks first if there is any data stored. Because if there is no data stored and the program attempts to remove a record, an error will generate.

```
  do {
    cout«"\nEnter the number of the entry that you want to delete
      \n(you can look up this number if you list the entries)
      \nPress 0 if you don't want to delete:";
    cin»Num;
    if(! ((Num>=0)&&(Num<X. lengthO)))
      cout«"\nTHIS ENTRY DOES NOT EXIST.\n\n";
  }while(! ((Num>=0)&&(Num~=X.lengthO)));
```

If the user enters a number for an entry to delete that does not exist, this part of the coding will prevent any errors from generating because it will ask the user again for the number of the entry.

This is a good approach but certainly more than half of such features should be documented in this way for an award of level 2.

Where there are very many examples of such error-handling that are all similar, the student could get away with describing the general process and then giving appropriate examples. However, in this case, perhaps a more general validation routine should be considered, as shown in the previous example.

Here is an example using a Word table.

Error Handling
This is most apparent in the BrainTest and BrainTestEdit Classes where helpful error messages are output.

These are listed below:

Screenshot 1
The DataHandler Class uses try – catch blocks extensively. Code that is liable to cause an i/o error is enclosed in a try block then if an error occurs control passes to the catch block, for example:

```
public void saveMusicQuestions(String[] questions)
{
  // overwrite the file contents with the contents of the array
  try
  {
    PrintWriter outText = new PrintWriter(new FileWriter(musicQFile));
     for(int x = 0; x < questions.length; x++)
    {
      outText.println(questions[x]);
    }
    outText.println(Parameters.EOF);
    outText.close();
  }
  catch(IOException ioe)
  {
      IoFlag = false;
      IoMessage = ioe.getMessage();
  }
}
```

The catch block is used to set the status of the IoFlag and IoMessage variables. After a read/write operation in DataHandler, these flags can be accessed by the calling routine:

```
private boolean loadQuestions()
{
  musicClip = theHandler.getMusicClip();
  introText = theHandler.getIntroText();
  testPanel.setIntro(introText);
  mathQ = theHandler.getMathQ();
  mathA = theHandler.getMathA();
  musicQ = theHandler.getMusicQ();
  // returns the status of the IoFlag, False if there was an error in
  // the file handling.
  return theHandler.getIoFlag();
}
```

Level 3

Achievement level	Descriptor
3	The student **fully** documents the error-handling of **each** input and output method within the program.

For an award of level 3 the documentation needs to be thorough and complete.

Criterion C3: Success of the program

Evidence here refers to hard copy output in criterion D1.

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student **includes** evidence that the program **functions partially**. The student successfully achieved **some** of the objectives from criterion A2.
2	The student includes evidence that the program functions **well**. The student successfully achieved **most** of the objectives from criterion A2.
3	The student includes evidence that the program functions well. The student successfully achieved **all** of the objectives from criterion A2.

The teacher should run the program with the student to confirm that the program functions, and that it produces the hard copy output submitted with the program dossier.

Level 0

Achievement level	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.

If no hard copy is presented at all for criterion D1, then 0 is also an appropriate award for this criterion, irrespective of the claims of the student and teacher as to the success of all or parts of the program.

One possible exception could be where the student has a goal that cannot be represented on hard copy output.

The testing system:

Objectives	**Evidence that it does**
Plays a set piece of music during the test.	It works but I can't give printed evidence.

Level 1

Achievement level	Descriptor
1	The student **includes** evidence that the program **functions partially**. The student successfully achieved **some** of the objectives from criterion A2.

Achievement or qualified achievement or even non-achievement should be stated in this section and backed up by reference to hard copy output.

Objectives	**Evidence that it does (see D1)**
Collects the relevant information from the test taker, this is: • name • sex • answers to maths questions - numeric • answers to music questions – true/false;	It does this. **Screenshot 1** **Screenshot 3** **Screenshot 6** **Screenshot 7**
Calculates: • the score • the time taken	**Screenshot 2**
Stores all of the above data in a csv file.	**Screenshot 14**

Level 2

Achievement level	Descriptor
2	The student includes evidence that the program functions **well**. The student successfully achieved **most** of the objectives from criterion A2.

To achieve this level, evidence that the program meets most of the objectives in full should be presented.

Level 3

Achievement level	Descriptor
3	The student includes evidence that the program functions well. The student successfully achieved **all** of the objectives from criterion A2.

As with some previous criteria, the award of a level 3 is a matter of degree. The teacher should be in a good position to judge this award by, say, sitting with the candidate and observing that the claimed objectives have been fulfilled successfully.

Stage D—Documentation

Criterion D1: Including an annotated hard copy of the test output

The hard copy of test output should demonstrate that the program fulfills the criteria for success in criterion A2. The output must be annotated (this may be done by hand). The teacher must confirm that each student has actually completed the testing as claimed in the documentation. (See the *Vade Mecum*.)

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student includes an **incomplete** set of sample output.
2	The student includes an incomplete set of **annotated** sample output.
3	The student includes a **mostly complete** set of annotated sample output.
4	The student includes a **complete** set of annotated sample output, testing all the objectives in criterion A2.

Hard copy output from one or more sample runs should be included to show that the different branches of the program have been tested; testing one set of valid data will not be sufficient. The hard copy submitted should demonstrate the program's responses to inappropriate or erroneous data, as well as to valid data. Thus the usefulness of the error-handling routines mentioned above should become evident. While at least one complete test run must be included in the dossier, it is not necessary that the hard copy reflect every key stroke of every test run. Cutting and pasting of additional test runs should be done to illustrate the testing of different aspects of the program.

All test runs should be annotated in such a way that the student is stating what aspect of the program is being tested. Sample output must **never** be altered by hand, erased or covered up.

Sample output can be "captured" and combined electronically with explanatory annotations into a single document. However, it is forbidden to alter or reformat sample output in any fashion (except to add page numbers or annotate in order to highlight user friendliness or error-handling facilities as discussed above), especially if these alterations would give an unrealistic impression of the performance of the program. Examples of such "abuse" include: lining up text that was not originally aligned; adding colour or other special effects; changing incorrect numerical output; erasing evidence of errors.

Level 0

Achievement level	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.

In this first example a student presents only screenshots with no annotations and no demonstration runs. This does not constitute a hard copy of test output.

Deleting a Player

```
Auto
                    1 - Add player
                    2 - Delete player
                    3 - Search for players by last name
                    4 - Display information of a player

                       Enter your option: 2

Enter the last name of the player you want to delete: Solorzano

PLAYER FOUND
Raul
Solorzano
Player's number: 45
Total shots: 9
Total shots made: 7
Total Number of Free Shots: 5
Total Number of Free Shots made: 3
Total Number of steals made: 2
Total Number of Turnovers made: 1
Total Number of offensive rebounds: 2
Total Number of defensive rebounds: 1
Total Number of fouls made: 3
Total Numer of assists made: 2

Do you wish to delete this PLAYER? (Y/N)
```

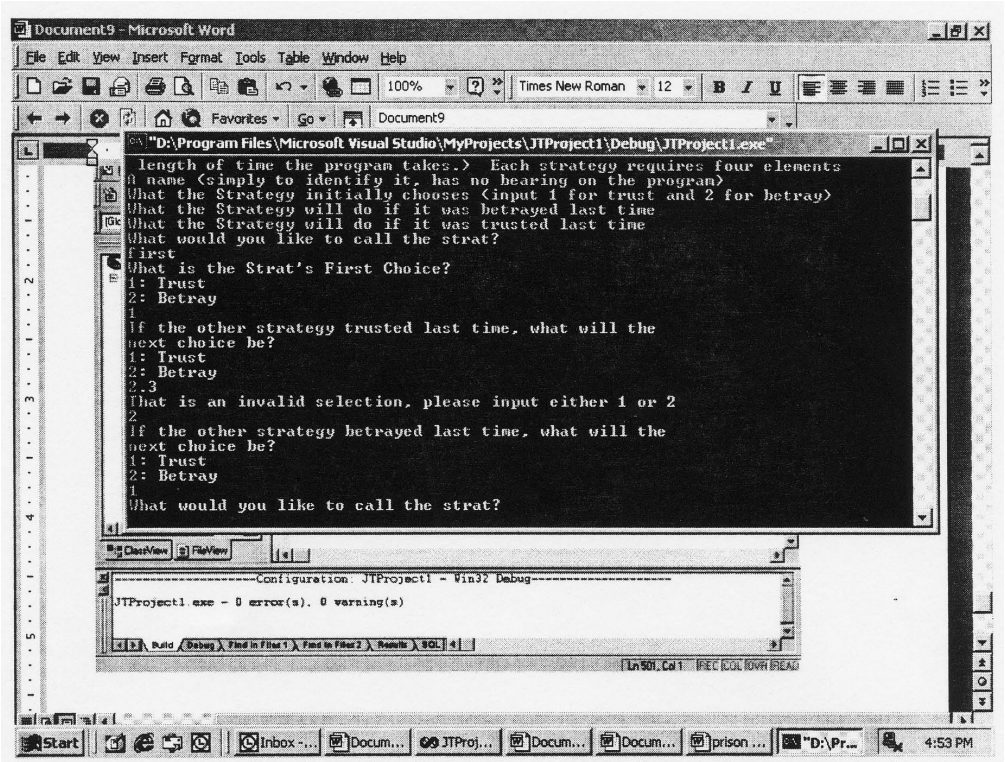
Searching for a Player

```
Auto
First Name: Raul
Player's number: 45
Total shots: 9
Total shots made: 7
Total Number of Free Shots: 5
Total Number of Free Shots made: 3
Total Number of steals made: 2
Total Number of Turnovers made: 1
Total Number of offensive rebounds: 2
Total Number of defensive rebounds: 1
Total Number of fouls made: 3
Total Numer of assists made: 2

                    BASKETBALL CAISSA DATABASE

                       Program Menu
                    0 - Exit program
                    1 - Add player
                    2 - Delete player
                    3 - Search for players by last name
                    4 - Display information of a player

                       Enter your option: 3

Enter the last name for which you want to search: Solorzano
```

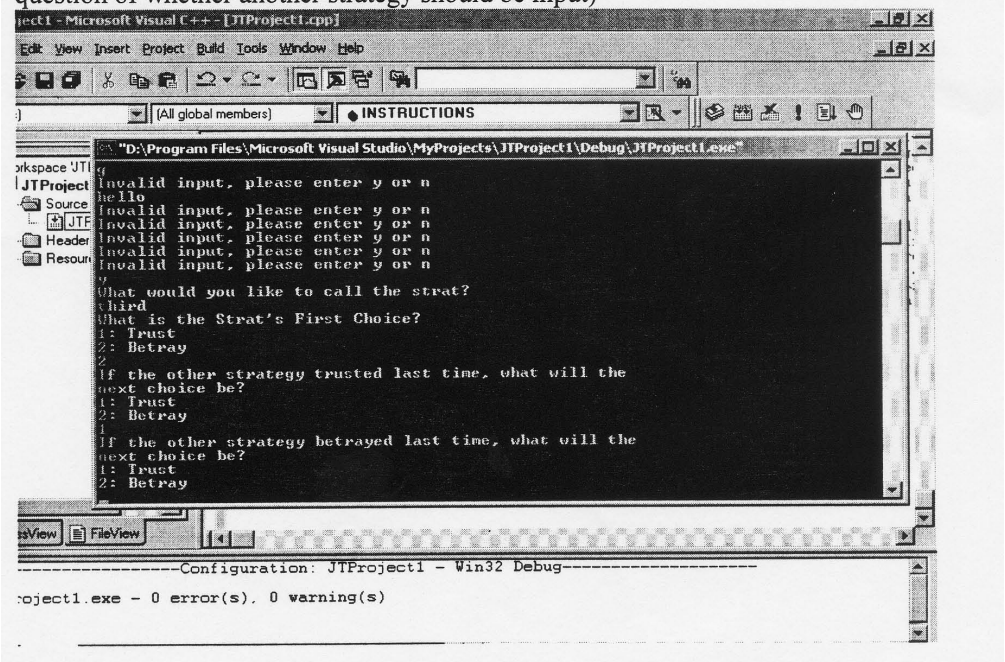
Level 1

Achievement level	Descriptor
1	The student includes an **incomplete** set of sample output.

Although the sample below is annotated (a level 2/3 requirement) and has some invalid data entered, it is one of only six screenshots and therefore the range of test runs is very limited.



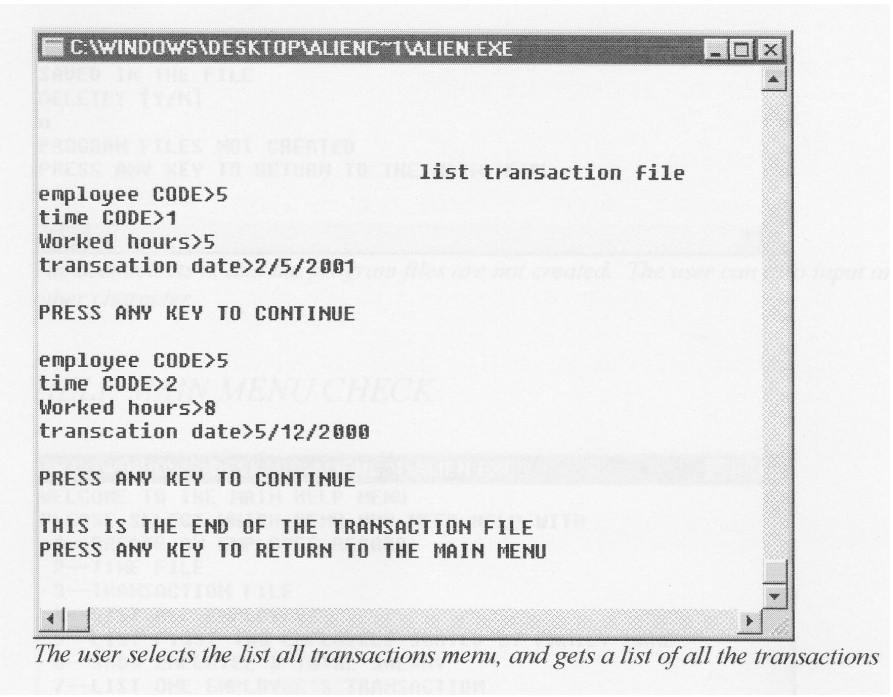
In the next screenshot, various illegal inputs appear near the top (in response to the question of whether another strategy should be input)



Level 2

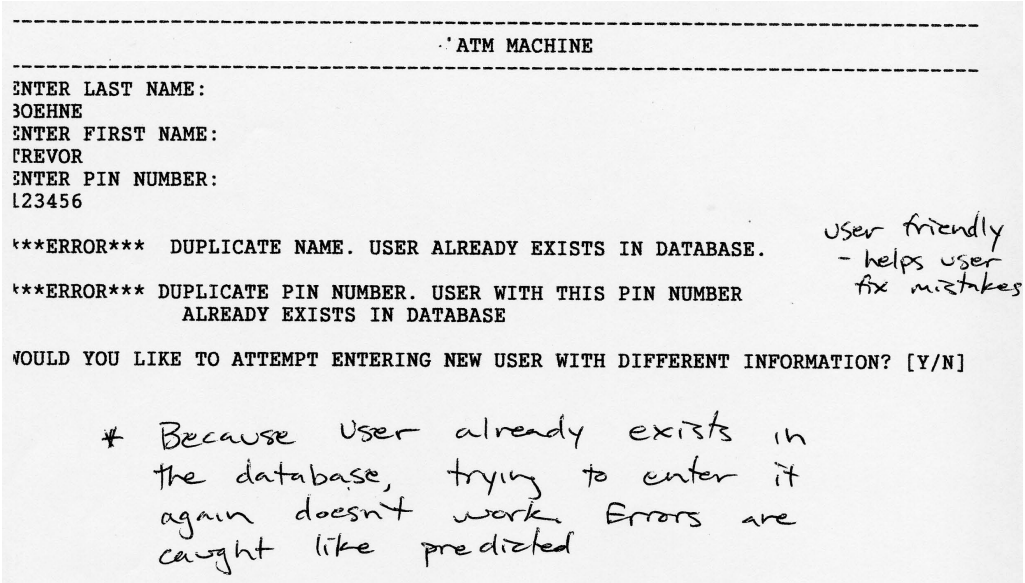
Achievement level	Descriptor
2	The student includes an incomplete set of **annotated** sample output.

This very good student has presented both invalid and valid data, but only a valid run is illustrated here. This run is annotated, which is a requirement for level 2.



The user selects the list all transactions menu, and gets a list of all the transactions

As this example shows, annotations can equally well be made by hand.



The mention of user friendliness supports awards for criterion C but should be a separate item and should be accompanied by the program code that produced the user-friendly feature.

Level 3

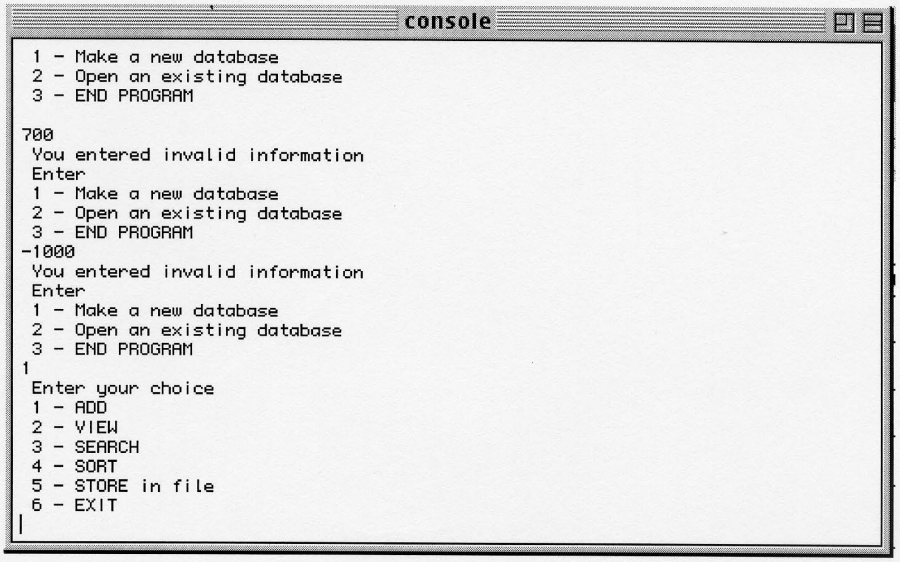
Achievement level	Descriptor
3	The student includes a **mostly complete** set of annotated sample output.

This excellent example, again only a small part of the total runs presented, is shown below. It also shows that the test plan has been carried out.

8. HARD COPY of testing

MAIN menu

Test for	Input	Expected output	Output
Low input	-1000	Error message	Error message
High input	700	Error message	Error message
	Return key	No changes	No changes
Data types	1a	Infinite loop	Infinite loop
Standard input	1	Menu1	Menu1



MENU1: Make a new database

Test for	Input	Expected output	Output
Low input	0	Error message	Error message
High input	1000000000	Error message	Error message
	Return key	No changes	No changes
Data types	1a	Infinite loop	Infinite loop
Standard input	4 (sort)	Sub-Menu1	Sub-Menu1

Level 4

Achievement level	Descriptor
4	The student includes a **complete** set of annotated sample output, testing all the objectives in criterion A2.

The award of a level 4 will depend to some extent on the criteria for success that were stated in section A2. These criteria should include the response of the program to valid and invalid input as these are a part of outlining "the limits under which the solution will operate".

This should be taken to include the stated requirement for functional testing as well as for data-entry testing (criterion A2 may have taken the form of a formal test plan, for example).

Criterion D2: Evaluating solutions

The evaluation/conclusion section should be a critical analysis of the resulting solution. Effectiveness should be discussed in relation to the original description of the problem and the criteria for success that were stated in criterion A2. Efficiency may be discussed in general terms, for example BigO notation is not required. Suggested improvements and possible extensions should be realistic, for example suggestions should not include statements such as "the program would be a lot better if it incorporated some artificial intelligence techniques such as speech recognition and natural language parsing".

Achievement levels	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.
1	The student only **outlines** the solution.
2	The student **outlines** the solution and **partly** considers effectiveness, efficiency and possible improvements.
3	The student **discusses** the effectiveness and efficiency of the solution **and suggests** alternative processes and improvements.
4	The student **suggests** alternative approaches to the solution and the design process.

This section of the dossier would typically be two pages in length.

The evaluation/conclusion should include reflections on the effectiveness of the programmed solution of the original problem. It should discuss answers to the following questions.

- Did it work?
- Did it address the criteria for success?
- Did it work for some data sets, but not others?
- Does the program in its current form have any limitations?
- What additional features could the program have?
- Was the initial design appropriate?

A thorough evaluation should also discuss possible future enhancements that could be made to the program.

Level 0

Achievement level	Descriptor
0	The student has not reached a standard described by any of the descriptors given below.

This student has included some conclusions but these do not really amount to an evaluation.

Conclusion

I think that with this program the problem was solved. It is a very easy to use program and in the future I will try to continue the program in order to be able to enter more information. I hope the people who use this program will find it useful as I did to have their contacts in a safe place.

The solution has not been outlined at all.

Level 1

Achievement level	Descriptor
1	The student only **outlines** the solution.

Conclusion

When evaluating my project, I must say that it was a very interesting experience. As I mentioned in the beginning I had three objectives:

1. Easy access
2. User friendly
3. Portability

Easy access-Yes, the program is easy to access by a icon placed on the desktop

User friendly-Not as simple as I wanted it to be, but it’s fairly easy to use

Portability-Not a problem, can be stored on a Floppy disk or a CD

As I stated in user friendliness, I was very much interested in peoples feedback. I also mentioned that I got a lot of positive feedback, but I was also told by some people that my address book was to simple for them and that I needed to add more features to it. I found these comments positive as well because it showed me that I has succeeded in creating a address book for the common computer user.

But not everything went perfect, I made a lot of mistakes while programming which could have been avoided if I would have been more organized. Java certainly was the right software to do this with, but my knowledge of Java wasn't as broad as it could have been, so the process of programming took longer than it could have taken. For anybody else trying to create an address book I can only give you the following advice. Keep checking if your programming works and make sure you check your system for errors all the time.

The student has considered the solution but not really outlined it as the comments could apply to any completed application. However, some thought has been given to issues, which allows the award of a (borderline) level 1.

The next example outlines the solution and problems encountered without considering other issues (the comments about graphical improvements do not really count).

Conclusion

Overall I was pretty satisfied with my program. However there were some little things I could have done in order for the program to more enhanced.

I was satisfied because this program really gave me the opportunity to explore my overall knowledge of Java. I was very successful in finishing it, without having many difficulties in doing it. The program is hundred percent robust, and works very good. Like you can see in my run section, it contains many user-friendly messages, and alerts.

The reason I wasn't satisfied with some of the aspects of my program is because I wasted way too much time on figuring out new commands I wanted to include, and therefore forgot to change some small errors within the program. For example I forgot to put percent signs next to the grades and similar small details like that. Another reason I wasn't satisfied with my program is because of the fact that I didn't include a Class Average Option, which I planed to do. I guess the problem must have been that I ran out of time.

Actually there not many unforeseen problems I encountered while creating the Teacher's Grade Book simulation. I finished my main required structure of the program in a very short time, however I had problems with my disk and lost almost half of it. That is mainly what slowed me down in finishing it, and adding new commands to it. Another thing I struggled with for a while was my average function. It kept giving me a syntactical error for a while, until I finally figured out that I had the variable I was dividing by, declared as a REAL, but the factor variable as an INTEGER, which obviously didn't work. Besides the two difficulties, I sincerely enjoyed creating this program and found it actually to be fairly easy.

Looking at my program right now, I guess I would want to add some graphics and sound effects to it. I would use graphics for drawing some type of background design. Sound effects would come in handy for creating warning sounds, as part of the user friendly messages and alerts.

From this program I gained a great amount of extra knowledge. Since we didn't discuss records and nested records that much in class, I did a lot of research of my own, which made it easier for me to comprehend records better. I feel like I've come a long way from when I first started programming. In just this last month I learned a whole lot, and it is much easier for me to create programs then when I first started getting aquatinted with the language.

Level 2

Achievement level	Descriptor
2	The student **outlines** the solution and **partly** considers effectiveness, efficiency and possible improvements.

This student gives a good account of the solution and certainly covers some relevant issues. However, it is questionable whether the part on efficiency qualifies as a discussion as it does not really address the efficiency of the solution but only compares it to that of the existing system.

Evaluation/Conclusion

In conclusion, this programmer believes that the proposed problems stated at the beginning of this dossier have been solved as initially planned. In the place of the inefficiency of the records previously maintained by restaurants (either by hand or on Microsoft Excel), this programmer has placed a simple easy to use Clientele database which allows for quick and easy data storage, recovery, and manipulation at fraction of the time, energy, and money. Instead of having a plethora of unnecessary functions at the fingertips of a computer illiterate worker with the potential to use any one of them to permanent damage the existing data, this programmer has placed an abstracted, simplified version with the latest in error management technology.

However, despite all these advantages, there are still quite a number of areas wherein this program can be improved; for either aesthetic or practical reasons. Aesthetically speaking, multi-colour displays could have been added to liven up the user-interface as well as the menu prompt. Different fonts and special icons always helps greatly in the work environment. Some of the more practical improvements that can be made include adding an interface between the program and a printer so as to be allow the person answering the phone call to attach the client's order to their record and print off a form straight off into the kitchen. Restaurant specifications can be modified into the program to allow the calculations of price from the menu of that restaurant which would then make it possible for the price to be printed onto the form as well.

All in all, I believe that this program is in good shape. Though there is much room for improvement, I believe that this program already greatly facilitates the day to day routines of the average food delivery service in a restaurant. Patches will be made and modifications will be coded continuously until finally even the most technologically adverse restaurants feel the need to use my product.

Level 3

Achievement level	Descriptor
3	The student **discusses** the effectiveness and efficiency of the solution **and suggests** alternative processes and improvements.

A good student will probably ensure that all parts of the criteria are at least attempted by using suitable sub-headings.

Evaluation of the Solution

Outline

The solution pretty much answers the problem set. It allows the system user to set up a maths test and set of music questions to be answered. These questions and the maths answers can be edited within the system or via a simple text editor such as Notepad since the decision to use text files was implemented.

From the test takers point of view it is a simple and easy to use application. The programming is modular and takes advantage of inheritance. It uses constants to set the major parameters of the application.

The code is relatively long but most of it is relatively simple. Searching and sorting were not implemented.

Effectiveness

The program fulfills all of the requirements initially set out with the exception of exporting data to Excel or Access. However this is not a serious limitation as the results file created is a csv text file which can be read/imported by these applications.

The application could handle the test taking and editing in other ways (eg via a text-based console system) but this is more user friendly and accessible for the system user and test taker.

One issue is the size of the application when it is distributed. A commercial system (Java Rollout) was used to prepare the file for running outside of the IDE. This system also detects if appropriate Java components are installed on the target machine and prompts the user to install, eg the Java Virtual Machine, if required. All of the data files and the wav file were then zipped up for distribution to end users (test takers and the system user). The total size of the zipped file was 3 Mb and this seems mainly due to the use of the Java Media Framework.

Efficiency

There are a few issues of efficiency but these do not seem to impact the running time significantly – which is fine. At times there is a delay in loading and playing the wav file when the maths test starts. Possibly this could be improved by using some kind of "pre-load" option in the JMF but was not critical in the wav file used – a 1.7 Mb file playing for 3-4 minutes.

Possibly the data loading and storage of results could have been handled with only one file – this would complicate the processing but reduce the chances of an i/o error causing early application exit.

The arrays of Strings did not take up a great deal of memory – this will not be a problem on any computer which meets the requirements for installation of the JRE and JMF components required for the application to run.

The application has modest storage requirements for data files – of the order of a few KB.

Possible improvements

The setting of application parameters in a separate Class was a good way to vary these if needed. An alternative approach would have been a configuration file, allowing the system user to control these parameters (as was initially proposed). However, this particular

application is a "one-off" for a specific purpose so does not need to be customized after the initial development is completed – therefore a configuration file was not really needed.

The JMF could have permitted more flexibility in the handling of media files – selecting, loading, looping and so on. However, again, this is not a Jukebox application so these were not needed. This is one reason why the JMF is really unsuitable – a sledgehammer approach.

The use of a configuration file would have allowed the varying of the number of questions should the system user require this.

There are a number of usability issues which could have been improved but these were not apparent from the non-functional prototype:

- The focus (cursor) should have been placed in the first text box when the screens were first loaded.
- The user should not have to press the next button after answering a maths question, pressing <enter> on the keyboard should have this effect.

Level 4

Achievement level	Descriptor
4	The student **suggests** alternative approaches to the solution and the design process.

Alternative approaches

The design methodology was adequate, not a thoroughly object oriented approach but more of an object-based modular one.

Some aspects of the initial design were not thoroughly thought out (eg, the data handling requirements). However, since these were put into a separate Class these could be worked through at the development stage. The modular design certainly made testing and debugging each unit an easy process.

The development of the prototype as an Applet helped greatly with the process, a PowerPoint presentation, for example, would not have allowed the experimentation with the Card Layout which was a very effective part of the solution.

There may be a better solution to the problem of playing the music but I wasn't able to identify it. Playing a wav file from an Applet is simple (and this would be ideal for the application as it could then be web-based). However the need to load and save data files means that an Application has to be used as ordinary Java Applets do not permit this. The JMF seems to be the only solution. Possibly a signed Applet could have been used as this does allow Applets to step out of "the sandbox" but it is doubtful that user's would want to accept an Applet that can access local system files unless it was from an established software company.

Stage E—Holistic approach

Criterion E: Holistic approach to the dossier

The program dossier should be an ongoing process involving consultation between the student and teacher. The student should be aware of the expectations of the teacher from the beginning of the process and each achievement level awarded should be justified by a written comment from the teacher at the time of marking. The examples given below for each criterion level are teacher-orientated and each teacher should use discretion when judging the levels.

Achievement levels	Descriptor
0	The student showed **no** commitment. For example, the student did not participate in class discussions on dossier work, did not submit the required work in progress, and/or missed many deadlines.
1	The student showed **minimal** commitment. For example, the student participated minimally in class discussions on dossier work, kept to most deadlines, had some discussion initiated by the teacher and/or did not exploit the available opportunities for the development or improvement of the dossier.
2	The student showed **good** commitment. For example, the student participated in class discussions on dossier work, initiated discussions with the teacher and/or the rest of the class and/or became fully involved in the development of the dossier.

In order to obtain the highest achievement level for this criterion the student should have excelled in areas such as those listed below. This list is not exhaustive and teachers are encouraged to add their own expectations.

The student:

- actively participated at all stages of the development of the dossier
- demonstrated a full understanding of the concepts associated with his/her dossier
- demonstrated initiative
- demonstrated perseverance
- showed insight
- prepared well to meet deadlines set by the teacher**.**

No sample material can be presented for this section as the awards are completely within the teacher's hands.

A possible example record sheet that might be used or adapted for use with this section is given here.

Holistic approach:

Aspect	Comment	Mark 0–2
Participation	Worked well with the rest of the class; made helpful suggestions to colleagues.	2
Understanding	Good.	2
Initiative	Often approached the teacher to discuss dossier issues; was always willing to try a different approach.	2
Perseverance	Could have thought more about some of the problems encountered before seeking assistance.	1
Insight	Good.	2
Deadlines	Missed one deadline out of the four set.	1

Additional comments:

Total awarded (max 2): 2