### Competition Problem Solving

<table>
<thead>
<tr>
<th>Designer(s)</th>
<th>Coder(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Design the Algorithm</strong>&lt;br&gt; This may be a full or partial algorithm, it should be in English or any type of pseudo-code.</td>
<td><strong>Input the Data</strong>&lt;br&gt; The first step is <em>always</em> to write code that reads in and outputs the data. The output may be commented out later, but unless you can check that you are reading the data in correctly you have problems.</td>
</tr>
<tr>
<td><strong>2 Test the Algorithm</strong>&lt;br&gt; Test the algorithm against the data provided as test data. This needs to be done as a desk check. There is no point giving the coder an algorithm that has not been shown to work. Pass the algorithm to the coder.</td>
<td><strong>Data Structures</strong>&lt;br&gt; Work out how to store any data that you need from that input. Page ?? has a discussion about data structures.</td>
</tr>
<tr>
<td><strong>3 Create More Test Data</strong>&lt;br&gt; Look for boundary conditions, unusual possibilities and normal probabilities. Work out input and output for these. Pass these new sets of test data on to the coder.</td>
<td><strong>Data Structures Again</strong>&lt;br&gt; Look at the algorithm provided. Work out how you are going to store data during the run of the program. Page ?? has a discussion about data structures.</td>
</tr>
<tr>
<td><strong>4 Start Designing the Next Solution</strong>&lt;br&gt; While the coder is getting the last problem done, you need to be getting ahead on the design. The design forms a greater and greater part of each problem. Two people can either work together on the design or work on different problems.</td>
<td><strong>Code the Provided Solution</strong>&lt;br&gt; Type in code to match the provided algorithm. If you don’t understand something query it, don’t just start coding something different. Remember to put in some comments and to refactor when necessary - it may not be you who debugs it in the end!</td>
</tr>
<tr>
<td><strong>5 Continue Designing</strong>&lt;br&gt; Keep designing: even if you are multiple problems ahead!</td>
<td><strong>Test the Solution</strong>&lt;br&gt; Make sure the algorithm is tested against both the test data provided <em>and</em> that worked out by the designers. If you cannot get it to work then you need to print your code and give it to one of the designers to find the discrepancy between their design and your code.</td>
</tr>
<tr>
<td><strong>6 Continue Designing</strong></td>
<td><strong>Submit Solution</strong>&lt;br&gt; Submit the solution and make sure you write down <em>exactly</em> what response you get.</td>
</tr>
<tr>
<td><strong>7 Submission Success or Failure</strong>&lt;br&gt; If the submission is successful then go back to Step 1 above. If the submission fails, then the problem needs to be solved - see the steps on page ??.&lt;br&gt; If solving the problem involves re-design, then while the designers are working on that, the coder should start coding the next problem, even if it is just input/output.</td>
<td></td>
</tr>
</tbody>
</table>
The Submission Failed

1. If a formatting error, the Coder must compare output with expected output, looking for spaces, line breaks etc. Fix and resubmit. On page ?? there are hints and information about output.

2. If a time-out error, the Coder must check that loops do finish.
   If they do, then the algorithm must be given back to one of the Designers so that they can work out a better solution. Make sure they get a printout of what was coded.

3. If an incorrect answer—assuming you have tested it with all available test data—then this means that the algorithm was incorrect for some possible data. In other words the designers made-up test data was lacking! The algorithm goes back to one of the Designers for checking with more—better—test data.

What You Need With You on the Day

1. At least 3 writing pads, multiple colours of pens, pencils, squared or columned paper for desk checks.

2. Multiple Books on:
   (a) Basic C++.
   (b) The Standard Template Library (STL).
   (c) Advanced C++ with Data Structures (graphs, linked lists, trees etc).
   (d) Algorithm Design.

   There are books in the library as well as on my bookshelf, so make sure you stock up in the weeks beforehand.

3. All the ICT209 lecture notes, plus the index to them.

4. Any ICT209 example code.

5. Every handout that I have given you.

6. Every solution from any source for past problems.

7. Multiple layers of clothing - the rooms tend to start cold, get warm as people and computers warm up and then get cold again when someone switches on the aircon.

8. Patience, brains, cooperation, faith and stamina!

9. You do not need to bring food, it is all supplied. However last year they forgot to supply ordinary water, so you might want to bring a bottle of that each (there is somewhere to fill it when empty).
What You Are Not Allowed on the Day

1. Disks, thumbdrives etc.
2. Mobile phones (give them to me).
3. Radios, mpeg players, walkmans etc

Data Structures

1. Make sure that you use the correct data type for the problem. If accuracy is important, use integers: floating point numbers cannot be stored accurately in a computer. For example if storing money, use integers, and work in cents.

2. If very large numbers are necessary then use floating point numbers, but bear in mind that you have an inaccuracy. And if converting to integer then make sure you know if you want the floor, ceiling or rounding of the floating point number! For example:
   \[
   \text{ceil}(2.4) = 3.0, \text{floor}(2.4) = 2
   \]
   To round a number, add 0.5 and take the floor:
   \[
   \text{floor}(\text{num} + 0.5)
   \]
   rounds it to the nearest integer.

3. For strings use the STL string. An example of this can be found on page ??.

4. Remember that structs and classes are very similar: you can use either in the competition. Generally if a set of data is going to need to be operated on then I use a class, otherwise I use a struct. Note that you do not need to keep data private for competitions. An example of each was given in Worksheet 3.

5. The most common more complicated data structures that you will need are:
   - An array - use the STL vector (see Worksheet 4 and page ??, below).
   - A set - this is a container of things of similar type. It is a mathematical concept and used to decide if there are intersections (etc) between the sets. Use the STL set (see page ?? below).
   - A stack - this is like a stack of books - what goes on first, comes off last - use the STL vector or deque (see page ?? below).
   - A queue - this is just like a queue at the movies - what joins the queue first, comes off first - use the STL deque (see page ?? below).
   - An association - this is like a word and its dictionary definition, it is used within a dictionary (see next point).
   - A dictionary - this is a set/array/container of associations - use the STL map (see page ?? below).
   - A two-dimensional structure. For this you can use any combination of arrays, stacks, queues, dictionaries as appropriate (see page ?? below).
Output Hints

1. New-fashioned C++ output makes some tasks easier because cout ‘knows’ the type it is trying to output. However I find formatting much more difficult. Make sure you always include <iomanip> so that you can use these functions. Examples of formatting are shown below:

   - Justification is set and then stays set until changed:
     To specify left justification:
     ```cpp
     cout.setf(ios::left)
     ```
     or right:
     ```cpp
     cout.setf(ios::right)
     ```
     to unset a justification:
     ```cpp
     cout.unsetf(ios::right)
     ```
   
   - Column widths have to be set individually for each item output:
     ```cpp
     cout << setw(8) << num1 << setw(8) << num2 << setw(20) << name << "\n";
     ```
   
   - Setting the column width will put the name in the example above right justified in the column. To make it left justified (as per normal), but then ensure all future output uses the default:
     ```cpp
     cout << setw(8) << num1 << setw(8) << num2 << " ";
     cout.setf(ios::left)
     cout << setw(20) << name << "\n";
     cout.unsetf(ios::left)
     ```
   
   - To add leading zeros to pad a number for the width of the column:
     ```cpp
     cout << setfill('0')
     ```
     and remember to turn it back to the default afterwards:
     ```cpp
     cout << setfill(' ')
     ```
   
   - To output in hexadecimal:
     ```cpp
     cout.setf(ios::hex, ios::basefield)
     ```
     and return it to normal afterwards:
     ```cpp
     cout.unsetf(ios::hex);
     ```
   
   - To output all floats in five decimal places until further notice:
     ```cpp
     cout << fixed << setprecision(5) << "\n";
     ```
     and return it to normal:
     ```cpp
     cout.unsetf(ios::fixed);
     ```
   
   - If you do not know a width then a variable can be used (note also the use of two lines):
     ```cpp
     cout << setw(maxWidth) << num1 << setw(maxWidth) << num2
     << setw(strWidth) << name << "\n";
     ```

2. Note that you can mix and match C++ and C input and output statements within one program with no problems.

cont...
3. If you want to use old-fashioned C output statements, that is fine (I do in competition stuff!). However you do need to make sure you tell `printf` what type you are outputting or you will get garbage out, *not an error message!*

Justification, column widths, leading 0s and type are all specified for each individual piece of output data. You need to include `<stdio>` and `<stdlib>`. Examples:

(a) A floating point number with the default decimal places:
```
printf ("%f", num);
```

(b) A floating point number on a line of its own:
```
printf ("%f\n", num);
```

(c) A floating point number with exactly 5 decimal places:
```
printf ("%.5f", num);
```

(d) A floating point number with exactly 4 decimal places in a column 8 wide:
```
printf ("%8.4f", num);
```

(e) An integer:
```
printf ("%d", num);
```

(f) An integer in a column 8 wide:
```
printf ("%8d", num);
```

(g) An integer in a column 6 wide, with leading 0s:
```
printf ("%06d", num);
```

(h) A hexadecimal integer:
```
printf ("%x", num);
```

(i) A pointer (address):
```
printf ("%p", num);
```

(j) A string:
```
printf ("%s", str);
```

(k) A string right justified in a column 20 wide:
```
printf ("%20s", str);
```

(l) A string left justified in a column 20 wide:
```
printf ("%-20s", str);
```

(m) A character as a character:
```
printf ("%c", ch);
```

(n) A character as its ASCII code:
```
printf ("%d", ch);
```

(o) Putting it all together (and note the use of two lines):
```
printf ("The character %c has the ASCII code %d, which is %x in hex\n", ch, ch, ch);
```

(p) Finally, if you do not know the width, a token can be used which is then replaced from the parameter list:
```
printf ("%*d %*.*f\n", width1, num1, width2, decPlaces, num2);
```
Strings

An example of string use:

```cpp
// Strings3.cpp
// Demonstrates the pre-defined STL string type
//--------------------------------------------------------
#include <iostream>
#include <iomanip>
#include <string>
#include <cstring>
#include <cstdio>
#include <cstdlib>
#include <cctype>
using namespace std;
//--------------------------------------------------------

int main()
{
    string str1 = "hello";
    string str2 = "world";

    cout << "str1 is '" << str1 << "' and str2 is '" << str2 << "' \n\n";

    cout << "Enter a new string for string 2: ";
    cin >> str2;
    cout << "str1 is '" << str1 << "' and str2 is '" << str2 << "' \n\n";

    str1 = str2 + " says hello";
    cout << "str1 is '" << str1 << "' and str2 is '" << str2 << "' \n\n";

    return 0;
}
```

A few tricks:

- Always include `<iostream>`, `<iomanip>`, `<string>`, `<cstring>`, `<cstdio>` and `<cstdlib>`. It doesn’t matter if they are included and not needed.
- Always put “using namespace std” underneath the includes.
- Any standard C function—such as `strcpy`—can be found in the man pages. Type: `man strcpy`.

cont...
• Conversion from lower case to upper case:
  \[ \text{ch} = \text{toupper}(	ext{ch}); \]
  and vice-versa:
  \[ \text{ch} = \text{tolower}(	ext{ch}); \]

• Conversion of an ascii digit (such as '8') to a number:
  \[ \text{num} = \text{ch} - '0'; \]
  and vice-versa:
  \[ \text{ch} = \text{num} + '0'; \]

• Getting the char* string out of an STL string:
  \[ \text{str.c_str();} \]

• Conversion of a char* string to an integer:
  \[ \text{int num = atoi(str);} \]
  or
  \[ \text{sscanf (str, "\%d", &num);} \]

• Conversion of a char* string to a floating point:
  \[ \text{float num = atof(str);} \]
  or
  \[ \text{sscanf (str, "\%f", &num);} \]

• Conversion of numbers to strings:
  \[ \text{sprintf (str, \"\%d\", \&num);} \]
  and
  \[ \text{sprintf (str, \"\%f\", \&num);} \]

• Copying a char* string to another char* string (but with no bounds checking!):
  \[ \text{strncpy(targetStr, sourceStr);} \]
  and to an STL string:
  \[ \text{targetSTLStr = sourceSTLStr;} \]

• Comparing two char* strings:
  \[ \text{if (strcmp(str1, str2) == 0)} \]
  and to an STL string:
  \[ \text{if (strcmp(str1, str2.c_str()) == 0)} \]
  and two STL strings:
  \[ \text{if (str1 == str2)} \]

• Reading a whole line, with spaces, into a char*:
  \[ \text{cin.getline(name, LINE_SIZE+1, '\n');} \]

• Testing for types of characters, these all return 0 if false, they require the <cctype> header file:
  \[ \text{isalpha(ch), isascii(ch), isdigit(ch), isalnum(ch), ispunct(ch), isspace(ch), isupper(ch), isxdigit()} \]

• If you need some kind of manipulation you don’t know about, then you can get information about most C type manipulation by typing “info libc” at the command line.
The STL, Iterators and Algorithms

Note: when using the STL, don’t forget to include the appropriate header file as well as the <algorithm> header file.

STL vectors (etc) can be accessed in two ways, by index and by iterator. Iterators are used by the STL methods for finding, deleting, sorting etc.

Note that all STL containers have an `empty()` function and most have a `size()` function.

```
// Define a vector of integers
typedef vector<int> IntVec;

// Declare a vector of integers
IntVec numbers;

// Put 100 random integers between 20 and 29 (inclusive) into the vector
for (int index = 0; index < 100; index++)
{
    int num = rand() % (30-20) + 20
    numbers.push_back(num);
}

// Output the integers using an index
int size = numbers.size();
for (int index = 0; index < size; index++)
{
    cout << numbers[index] << "\n";
}

// Output the integers using an iterator
IntVec::iterator itr;
for (itr = numbers.begin(); itr != numbers.end(); itr++)
{
    cout << *itr << "\n";
}

// And if you want the array sorted:
sort(numbers.begin(), numbers.end());

// Finding the first occurrence of 25 in the vector
IntVec::iterator itr;
itr = find(numbers.begin(), numbers.end(), 25);
cout << *itr;
```
// Deleting all integers after the first occurrence of 25 in the vector

IntVec::iterator itr;
itr = find(numbers.begin(), numbers.end(), 25);
numbers.erase(itr, numbers.end());

// Copying all integers after the first occurrence of 25 to the
// end of another vector

IntVec::iterator itr;
itr = find(numbers.begin(), numbers.end(), 25);
vector2.insert(vector2.end(),itr, numbers.end());

// Deleting all integers equal to 25 - method 1

IntVec::iterator itr;
for (itr = numbers.begin(); itr != numbers.end();)
{
    if (*itr == 25)
    {
        itr = numbers.erase(itr);
    }
    else
    {
        itr++;
    }
}

// Deleting all integers equal to 25 - method 2

IntVec::iterator itr;
itr = find(numbers.begin(), numbers.end(), 25);
while (itr != numbers.end())
{
    numbers.erase(itr);
    itr = find(numbers.begin(), numbers.end(), 25);
}
Sets

Sets are simply groups of similar items. The most common use is when you want to find the intersection between two sets of data. An example of sets and their algorithms:

```cpp
#include <iostream>
#include <set>
#include <algorithm>
#include <string>
#include <ctime>

using namespace std;
const int max = 30;
typedef set<int> IntSet;

void RandomFill (IntSet &aset);
void Output (char *label, const IntSet &set);

int main()
{
    // seed the random number generator
    srand (time(NULL));

    IntSet set1;
    IntSet set2;

    RandomFill (set1);
    RandomFill (set2);

    IntSet theIntersection, theDifference, theUnion;
    insert_iterator<IntSet> itr1 (theIntersection, theIntersection.begin());
    insert_iterator<IntSet> itr2 (theDifference, theDifference.begin());
    insert_iterator<IntSet> itr3 (theUnion, theUnion.begin());

    set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), itr1);
    set_difference(set1.begin(), set1.end(), set2.begin(), set2.end(), itr2);
    set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), itr3);
```
void RandomFill (IntSet &aset)
{
    int number = rand() % 30 + 1;
    for (int index = 0; index < number; index++)
    {
        int num = rand() % 30;
        aset.insert (num);
    }
}

//----------------------------------------
void Output (char *label, const IntSet &aset)
{
    cout << label << " : ";
    IntSet::iterator itr;
    for (itr = aset.begin(); itr != aset.end(); itr++)
    {
        cout << *itr << " ";
    }
    cout << "\n";
}
Stacks

• Stacks are First In Last Out (FILO) data structures.

• They are used for temporary storage during processing, where the order needs to be reversed from the original order, or where you can only access things at the top of the stack.

• Stacks can be implemented with either an STL vector or STL deque (some compilers have an STL stack also):
  
  ```cpp
  vector<int> stack;
  deque<int> stack;
  ```

• Data is pushed onto the end (top) of the stack:
  
  ```cpp
  stack.push_back(num);
  ```

• Data is removed from the end (top) of the stack:
  
  ```cpp
  num = stack[stack.size()-1];
  stack.pop_back();
  ```

• Commonly processing will continue until the stack is empty:
  
  ```cpp
  // Some data is added to the stack to initialise it
  // then processing can begin
  while (stack.size() != 0)
  {
    // Remove the data currently at the top of the stack
    int num = stack[stack.size()-1];
    stack.pop_back();

    // Something is done with num that results in more
    // things being pushed onto the end/top of the stack
  }
  ```
Queues

- Queues are First In First Out (FIFO) data structures.
- They are used for temporary storage during processing, where the order needs to be maintained from the original order.
- Queues are implemented with an STL deque:
  
  ```cpp
deque<string> queue;
  ```
- Data is pushed onto the end of the queue:
  ```cpp
queue.push_back(str);
  ```
- Data is removed from the beginning of the queue:
  ```cpp
str = queue[0];
queue.pop_front();
  ```
- Commonly processing will continue until the queue is empty:

```
// Some data is added to the queue to initialise it
// then processing can begin
while (queue.size() != 0)
{
    // Remove the item at the beginning of the queue
    string str = queue[0];
    queue.pop_front();

    // Something is done with str that results in more
    // things being added to the end of the queue
}
```
Associations and Maps

- Associations are paired pieces of information. For example a word and its dictionary definition, or a letter and its vertical mirror image:
  A:
a:B
b:p
e tc.
  There is a mapping from one part of the pair to the other.

- Dictionaries or maps are sets of associations (pairs).

- They are used to store data for easy lookup purposes.

- Dictionaries are implemented with an STL map:
  
  ```
  map<string,string> englishDictionary;
  map<char,char> verticalMirrorImages;
  map<int,char> keyboardCodes;
  ```

- Data is added to a map:
  
  ```
  verticalMirrorImages['A'] = ' ';
  verticalMirrorImages['a'] = ' ';
  verticalMirrorImages['B'] = 'B';
  verticalMirrorImages['b'] = 'p';
  ```
  etc.

- Data is found in a map using [] but with a first pair member, not an index:
  
  ```
  cout << "Enter a character: ";
  cin >> ch;
  if (verticalMirrorImages[ch] != ' ')
    
    cout << "Vertical mirror image of " << ch << " is" << verticalMirrorImages[ch] << "\n";
  
  else
    
    cout << "No vertical mirror image for " << ch << "\n";
  ```
Two Dimensions

- There are lots of ways to handle two dimensional structures.
- Essentially you just need to combine two one dimensional ones!
- For example, let's suppose you wanted to emulate a row of children's blocks with letters on them. Each block may have others piled on top of it, and of course they are stacked randomly:

```
R  W
O A E D
O O A F R
B V P Z O
B O D L E W I R
H Z C E Y P J A
P V V H T K R L J X
```

```cpp
// BlockPiles.cpp
#include <iostream>
#include <vector>
#include <ctime>
using namespace std;

// Define a type for a pile of blocks with letters on them
typedef vector<char> BlockPile;

// Define a row of these piles
typedef vector<BlockPile> PileRow;

// Prototypes of functions
void CreatePiles(PileRow &pileRow);
int FindHighest(const PileRow &pileRow);
void OutputPiles(int max, const PileRow &pileRow);

const int ROW_SIZE = 10;

//----------------------------------------------------------
```
```c
int main()
{
    // Randomise the number generator
    srand(time(NULL));

    // Declare a row of piles:
    PileRow pileRow;

    CreatePiles(pileRow);
    int max = FindHighest(pileRow);
    OutputPiles(max, pileRow);

    return 0;
}

void CreatePiles(PileRow &pileRow)
{
    // Put eight piles in this row, each with a random number of blocks in it,
    // less than 10
    for (int rindex = 0; rindex < ROW_SIZE; rindex++)
    {
        BlockPile pile;
        int numBlocks = rand() % 10;
        for (int pindex = 0; pindex < numBlocks; pindex++)
        {
            // Create a random character
            char ch = rand() % 26 + 'A';
            pile.push_back(ch);
        }
        pileRow.push_back(pile);
    }
}
```
int FindHighest(const PileRow &pileRow)
{
    // Find the largest number of blocks in a pile
    int max = 0;
    for (int rindex = 0; rindex < ROW_SIZE; rindex++)
    {
        if (pileRow[rindex].size() > max)
        {
            max = pileRow[rindex].size();
        }
    }
    return max;
}

void OutputPiles(int max, const PileRow &pileRow)
{
    int currentHeight = max;
    while (currentHeight > 0)
    {
        for (int rindex = 0; rindex < ROW_SIZE; rindex++)
        {
            if (pileRow[rindex].size() >= currentHeight)
            {
                cout << pileRow[rindex][currentHeight-1] << " ";
            }
            else
            {
                cout << " ";
            }
        }
        cout << "\n";
        currentHeight--;
    }
}
Some Common Programming Tasks

All of these use the definition `set<int> IntSet`.

1. Finding the factors of a number (i.e. all the numbers that divide into a specific number)

```cpp
void Factors (int num, IntSet &factors)
{
    // No number larger than half the number can divide into it,
    for (int index = 2; index <= num/2; index++)
    {
        // It divides exactly if there are no leftovers
        if ((num % index) == 0)
        {
            factors.insert(index);
        }
    }
    // except of course the number itself
    factors.insert(num);
}
```

2. Finding the common factors between two numbers:

```cpp
void CommonFactors (int num1, int num2, IntSet &commonFactors)
{
    // Find the factors
    IntSet factors1, factors2;
    Factors (num1, factors1);
    Factors (num2, factors2);

    // Find the intersection to get the common ones
    insert_iterator <IntSet> itr (commonFactors, commonFactors.begin());
    set_intersection(factors1.begin(), factors1.end(),
                    factors2.begin(), factors2.end(), itr);
}
```
3. Finding the highest common factor between two numbers:

```c
int HighestCommonFactor (int num1, int num2)
{
    // Find the common factors
    IntSet commonFactors;
    CommonFactors (num1, num2, commonFactors);

    // Return the maximum element of the set of common factors
    return *max_element(commonFactors.begin(), commonFactors.end());
}
```

4. Finding the lowest common denominator of two numbers:

```c
int LowestCommonDenominator (int num1, int num2)
{
    // Find the highest common factor
    int maxCommonFactor = HighestCommonFactor(num1, num2);

    // E.G. for 8 and 12 the HCF is 4, so the LCD = 4 * 8/4 * 12/4 = 24
    return maxCommonFactor * num1 / maxCommonFactor * num2 / maxCommonFactor;
}