CIL 2009 • Epreuve de Finale - Solutions proposées par Christian Kauth

Problème I - EXCLUSION — 20 points

```
/* solution by Christian Kauth
 *
 * ANALYSIS :  instead of testing all integer number in the interval [P;Q] against the 4 conditions
 *                      and eliminating the ones that do not respect all the conditions, we can tackle the
 *                      problem the other way round: We generate exclusively the numbers that will veryfy the
 *                      conditions. (for fun ;))
 */

#include <fstream>
#include <queue>
using namespace std;

#define MAX_NB 32767

bool isPrime[MAX_NB+1];
int P, Q, M, N;

// Eratosthenes filter
void primes()
{
      int i,j;
      isPrime[1] = false;
      isPrime[2] = true;

      for (i=3; i <= Q; i+=2) isPrime[i] = true;
      for (i=4; i <= Q; i+=2) isPrime[i] = false;

      for (i=3; i <= Q; i+=2)
            if (isPrime[i])
                  for (j=3; j*i <= Q; j+=2)
                        isPrime[i*j] = false;
}

void read_input()
{
      ifstream fin("EXCLUIN.TXT");
      fin>>P>>Q>>M>>N;
      fin.close();
}
```

```
void generate_valid_numbers()
{
	struct Candidate
	{
		int number;
		int dSum;
		Candidate(int n, int d) : number(n), dSum(d) {}
	};

	queue<Candidate> candidates;
	int number, dSum, newNumber, newdSum;
	ofstream fout("EXCLUOUT.TXT");

	candidates.push(*(new Candidate(0,0)));

	while (!candidates.empty())
	{
		number	= candidates.front().number;
		dSum  = candidates.front().dSum;
		candidates.pop();

		// conditions		c.)		d.)		a.)
		if ((number >= P) && (!isPrime[number]) && (!isPrime[dSum]) && (number%M) && (number%N))
			fout<<number<<" ";

		for (int d=0; d <= 9; d++)
		{
			newdSum= dSum+d;
			if (newdSum == 0) continue;		// the sum of zeros is a dangerous number ;)
			if ((d==M)|| (d==N)) continue;		// condition b.)
			newNumber = number*10+d;
			if (newNumber > Q) break;		// all further numbers will be out of the interval [P;Q]
			candidates.push(*(new Candidate(newNumber,newdSum)));
		}
	}
	fout<<endl;
	fout.close();
}

int main()
{
	read_input();
	primes();
	generate_valid_numbers();
	return 0;
}
```

Problème II - CODAGE 30 points

```
// solution by Christian Kauth

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

#define NCHAR     256               // number of lower case letters in the alphabet
#define MAX_SIZE  255         // maximum length of message

char encode[NCHAR];                 // map for encoding the characters
char decode[NCHAR];                 // map for decoding the characters
char message[MAX_SIZE];       // the message to be en- or decoded

bool read_input()
{
      string key;
      int e('a'), d('z');
      bool inputCorrect(true);
      bool used[26] = {0};
      ifstream fin("CODEIN.TXT");

      fin>>key;
      fin.getline(message,MAX_SIZE);
      fin.getline(message,MAX_SIZE);
      fin.close();

  // start with none of the ASCII characters encoded
      for (int i=0; i < NCHAR; i++)
      {
            encode[i] = i;
            decode[i] = i;
      }

  // use the key for encoding
      for (unsigned int i=0; i < key.size(); i++)
            if (!used[key[i]-97])
            {
                  decode[key[i]] = e;
                  encode[e] = key[i];
                  used[key[i]-97] = true;
                  e++;
            }
            else
                  inputCorrect = false;
```

```
  // use the remaining characters for encoding
      while (e<='z')
      {
            while (used[d-97]) d--;
            encode[e] = d;
            decode[d] = e;
            e++; d--;
      }

      return inputCorrect;
}

string encode_message()
{
      string s("");
      for (unsigned int i=0; i < MAX_SIZE; i++)
            s += encode[message[i]];
      return s;
}

string decode_message()
{
      string s("");
      for (unsigned int i=0; i < MAX_SIZE; i++)
            s += decode[message[i]];
      return s;
}

int main()
{
      char dir;
      ofstream fout("CODEOUT.TXT");

      cin>>dir;
      if (read_input())
            switch (dir)
            {
                  case 'C' : fout<<encode_message(); break;
                  case 'D' : fout<<decode_message(); break;
            }
      else
            fout<<"Mot secret non-valide"<<endl;

      fout.close();
      return 0;
```

Problème III - SOLITAIRE — 50 points

```
/* solution by Christian Kauth
 *
 * ANALYSIS :  The board has 15 cells and each cell can contain a draughtsman or not,
 *                meaning that only 2^15 different constellations exist. By making moves
 *                (one draughtsman jumps over another one, the latter being removed), we
 *                can switch from one constellation to another. After each move, there will
 *                be one draughtsman less on the board. Thus exactly 13 moves are needed.
 *                Our goal is to choose these 13 moves carefully, so that we reach the final
 *                constellation, starting from the initial constellation. For finding the
 *                sequence of moves, we use Breadth-First Search (BFS)
 *
 * ENCODING :  Next we need to encode the constellations in a meaningful way. As each cell
 *                can have only 2 states and there are 15 cells, I suggest encoding constellations
 *                as integers, each bit representing one cell. If the cell contains a
 *                draughtsman, the value of that bit is '1', otherwise it is '0'. We will assign
 *                the bit indexes as follows.
 *                                        0
 *                                    1       2
 *                                3       4       5
 *                            6       7       8       9
 *                        10      11      12      13      14
 *                Taking the example of the problemset, the initial constellation's code is
 *                0b111101111111111 = 31'743 and the final constellation's code 0b000010000000000
 *                = 1'024.
 *
 * REVERSE :   At the end, we need to output the sequence of moves that lead from the
 *                initial constellation to the final one. To avoid reversing the sequence of moves,
 *                we will simply start our BFS at the final constellation and try to recover the
 *                initial one, so that the output of the moves is straightforward.
*/

#include <fstream>
#include <vector>
#include <string>
using namespace std;

#define LENGTH          5                       // side length of the triangular board (try putting 6 here)
#define NCELLS          LENGTH*(LENGTH+1)/2     // number of cells on the board
#define NTURNS          NCELLS-2                // the number of moves that separate the initial from the final constellation
#define UNDEFINED       0                       // initialization purpose


struct Move
{
      unsigned int startPos;                    // bit index of the jumping draughtsman
```

```
	unsigned int jumpPos;			// bit index of the over-jumped draughtsman
	unsigned int landPos;			// bit index of the landing cell of the jumping draughtsman
	string name;				// the name of the jump (e.g. "3 1 -> 5 1")
	Move() {}
	Move(unsigned int s, unsigned int j, unsigned int l, string n) : startPos(s), jumpPos(j), landPos(l), name(n) {}
};

int bit[LENGTH][LENGTH];		// map indicating the index of the bit of a given cell (e.g. bit[5][1] = 10)
vector<Move> moves;			// collection of all the possible moves
unsigned int finalPos;		// the code of the final constellation
unsigned int nextConstellation[1<<NCELLS] = {UNDEFINED};	// initConst -> nextConsteallation[initConst] ->
nextConstellation[nextConsteallation[initConst]] -> .. -> finalPos
unsigned int nextMove[1<<NCELLS] = {UNDEFINED};		// the move to take at each constellation to reach the final one

// compute the bit index of each cell
void bit_indexes()
{
	int index(0);
	for (int r=0; r < LENGTH; r++)
		for (int c=0; c <= r; c++)
			bit[r][c] = index++;
}

// compute the names of the moves
string to_string(int r1, int c1, int r2, int c2)
{
	string s("");
	s += char(r1+49);
	s += " ";
	s += char(c1+49);
	s += " -> ";
	s += char(r2+49);
	s += " ";
	s += char(c2+49);
	return s;
}

// define all the possible moves
void define_moves()
{
	moves.push_back(*(new Move(0,0,0,"This is not a move")));  // placeholder

	for (int r=0; r < LENGTH; r++)
		for (int c=0; c <= r; c++)
		{
			if (c < r-1)			// horizontal jumps
			{
```

```
                        moves.push_back(*(new Move(1<<bit[r][c],1<<bit[r][c+1],1<<bit[r][c+2],to_string(r,c,r,c+2))));
                        moves.push_back(*(new Move(1<<bit[r][c+2],1<<bit[r][c+1],1<<bit[r][c],to_string(r,c+2,r,c))));
                  }
                  if (r < LENGTH-2)       // diagonal jumps
                  {
                        moves.push_back(*(new Move(1<<bit[r][c],1<<bit[r+1][c],1<<bit[r+2][c],to_string(r,c,r+2,c))));
                        moves.push_back(*(new Move(1<<bit[r+2][c],1<<bit[r+1][c],1<<bit[r][c],to_string(r+2,c,r,c))));
                        moves.push_back(*(new Move(1<<bit[r][c],1<<bit[r+1][c+1],1<<bit[r+2][c+2],to_string(r,c,r+2,c+2))));
                        moves.push_back(*(new Move(1<<bit[r+2][c+2],1<<bit[r+1][c+1],1<<bit[r][c],to_string(r+2,c+2,r,c))));
                  }
            }
}

void read_input()
{
      int r, c;
      ifstream fin("SOLITIN.TXT");
      fin>>r>>c;
      fin.close();
      finalPos = 1<<bit[r-1][c-1];
}

void play_solitaire()
{
      vector<unsigned int> next[2];
      bool ok(true);
      int prevConstellation;

      next[ok].push_back(finalPos);

      // BFS
      for (int t=0; t < NTURNS; t++)
      {
            for (unsigned int c=0; c < next[ok].size(); c++)
                  for (unsigned int m=1; m < moves.size(); m++)
                        // remember, we play the game backwards: start and jump positions must be empty and a draughtsman must be at the landing position
                        if ((next[ok][c] & moves[m].landPos) && !(next[ok][c] & moves[m].jumpPos) && !(next[ok][c] & moves[m].startPos))
                        {
                              prevConstellation = (next[ok][c] | moves[m].jumpPos | moves[m].startPos) ^ moves[m].landPos;
                              if (nextConstellation[prevConstellation] == UNDEFINED)
                              {
                                    nextConstellation[prevConstellation] = next[ok][c];
                                    nextMove[prevConstellation] = m;
                                    next[!ok].push_back(prevConstellation);
                              }
                        }
            next[ok].clear();
```

```
            ok = !ok;
      }
}

void output_moves()
{
      int constellation (finalPos ^ ((1<<NCELLS)-1)); // = initial position
      ofstream fout("SOLITOUT.TXT");

      if (nextMove[constellation] == UNDEFINED)
            fout<<"Pas de solution"<<endl;
      else
            for (int t=0; t < NTURNS; t++)
            {
                  fout<<moves[nextMove[constellation]].name<<endl;
                  constellation = nextConstellation[constellation];
            }
      fout.close();
}

int main()
{
      bit_indexes();
      define_moves();
      read_input();
      play_solitaire();
      output_moves();
      return 0;
}
```