Solutions écrites en C++ par l'élève Ben Strasser

I. Superdiv

```
#include<vector>
#include<iostream>
using namespace std;

typedef unsigned uint;

uint get_num(const vector<uint>&in){
	uint sum = 0;
	for(uint i=0,size=in.size(); i<size; ++i){
		sum *= 10;
		sum += in[i];
	}
	return sum;
}

bool contains(const vector<uint>&in, uint dig){
	for(uint i=0,size=in.size(); i<size; ++i)
		if(in[i] == dig)
			return true;
	return false;
}

void add_digit(vector<vector<uint> >&pos)
{
	vector<vector<uint> >new_pos;
	for(uint i=0, size = pos.size(); i<size ;++i){
		uint num = get_num(pos[i]) * 10;
		uint div = pos[i].size() + 1;
		for(uint digit=1; digit<=9; ++digit){
			if((num + digit) % div == 0){
				if(!contains(pos[i], digit)){
					vector<uint>new_num(pos[i]);
					new_num.push_back(digit);
					new_pos.push_back(new_num);
				}
			}
		}
	}
	pos.swap(new_pos);
}

void init(vector<vector<uint> >&in){
	in.clear();
	for(uint i=1; i<=9; ++i){
		vector<uint>num(1);
		num[0] = i;
		in.push_back(num);
	}
```

```
}

void print(ostream&out, vector<vector<uint> >&pos){
	for(uint i=0, size=pos.size(); i<size; ++i)
		out<<get_num(pos[i])<<endl;
}

int main()
{
	vector<vector<uint> >pos;
	init(pos); // add first digit
	for(uint i=2; i<=9; ++i){
		add_digit(pos);
	}
	print(cout, pos);
	cin.get();
}
```

II. LookAndSay

```
#include<fstream>
#include<iostream>
#include<string>
#include<sstream>
using namespace std;
typedef unsigned uint;

string gen_next(const string&in)
{
	ostringstream out;
	for(uint i=0,size = in.size(); i<size;)
	{
		uint count = 1;
		char what = in[i];
		++i;
		while(i<size && in[i] == what){
			++i;
			++count;
		}
		out<<count<<what;
	}
	return out.str();
}

int main()
{
	string str = "1";
	ofstream out("LookAndSay.TXT");
	uint lines;
	cin>>lines;
	for(uint line=0; line<lines; ++line){
		out<<str<<endl;
		str = gen_next(str);
	}
}
```

III. Sudoku

```
#include<fstream>
```

```
#include<iostream>
#include<vector>
using namespace std;

class no_solution{};

typedef unsigned uint;

// 1 possible
// 0 not possible


typedef std::vector<uint> Board;

class Line{
public:
	Line(Board&board, uint line):
		board(board), line(line){}

	uint&operator()(uint col){
		return board[line*9 + col];
	}
private:
	Board&board;
	uint line;
};

class Col{
public:
	Col(Board&board, uint col):
		board(board), col(col){}

	uint&operator()(uint line){
		return board[line*9 + col];
	}
private:
	Board&board;
	uint col;

};

class Box{
public:
	Box(Board&board, uint box):
		board(board), x(box/3), y(box%3){}
	Box(Board&board, uint x, uint y):
		board(board), x(x), y(y){}

	uint&operator()(uint n){
		return board[(x*3 + n/3)*9 + (y*3 + n%3)];
	}
private:
	Board&board;
	uint x,y;
};

void readin(istream&in, Board&board){
	for(uint y=0; y<9; ++y){
		for(uint x=0; x<9; ++x){
			uint num;
			in>>num;
			if(num == 0)
				board[x*9+y] = 0x1FF;
			else
```

```
                        board[x*9+y] = 1<<(num-1);
                }
        }
}

uint get_val(uint cell)
{
        uint val = 0;
        for(uint i=0; i<9; ++i){
                if(cell & (1<<i)){
                        if(val == 0)
                                val = i+1;
                        else
                                return 0;
                }
        }
        if(val == 0)
                throw no_solution();
        return val;
}

void writeout(ostream&out, Board&board){
        for(uint y=0; y<9; ++y){
                for(uint x=0; x<9; ++x){
                        out<<get_val(board[x*9 + y])<<" ";
                }
                out<<endl;
        }
}

template<class Seq>
bool exclude(Seq seq){
        bool worked = false;
        for(uint i=0; i<9; ++i){
                uint val = get_val(seq(i));
                if(val != 0){
                        val = ~(1<<(val-1));
                        for(uint j=0;j<9; ++j)
                                if(i != j){
                                        uint pre = seq(j);
                                        seq(j) &= val;
                                        if(seq(j) != pre)
                                                worked = true;
                                }
                }
        }
        return worked;
}

bool exclude_all(Board&board){
        bool worked = false;
        bool worked_now;
        do{
                worked_now = false;
                for(uint i=0;i<9;++i){
                        if(exclude(Line(board, i)))
                                worked_now = true;
                        if(exclude(Col(board, i)))
                                worked_now = true;
                        if(exclude(Box(board, i)))
                                worked_now = true;
                }
                if(worked_now)
                                worked = true;
```

```
	}while(worked_now);
	return worked;
}

// if only one possible
template<class Seq>
bool fill(Seq seq){

	bool worked = false;
	for(uint num =0; num<9;++num)
	{
		uint where = 0;
		for(uint i=0; i<9; ++i){
			if(seq(i) & (1<<num)){
				if(where == 0)
					where = i + 1;
				else
					goto long_break;
			}
		}
		if(where != 0)
		{
			if(seq(where-1) != (1u<<num))
			{
				seq(where-1) = (1u<<num);
				worked = true;
			}
		}
	long_break:;
	}
	return worked;
}

bool fill_all(Board&board){
	bool worked = false;
	bool worked_now;
	do{
		worked_now = false;
		for(uint i=0;i<9;++i){
			if(fill(Line(board, i)))
				worked_now = true;
			if(fill(Col(board, i)))
				worked_now = true;
			if(fill(Box(board, i)))
				worked_now = true;
		}
		if(worked_now)
				worked = true;
	}while(worked_now);
	return worked;
}

bool is_solveed(Board&board)
{
	for(uint i=0; i<9*9; ++i)
		if(get_val(board[i]) == 0)
			return false;
	return true;
}


uint find_best_guess(Board&board){
	uint best = 0;
	uint best_bit = 50;
```

```
	for(uint i=0; i<9*9; ++i)
	{
		uint bit_count = 0;
		for(uint j=0; j<9; ++j)
			if(board[i] & (1<<j))
				++bit_count;
		if(bit_count<best_bit && bit_count != 1){
			best = i;
			best_bit = bit_count;
		}
	}
	return best;
}

bool solve(Board&board){
	do{
		exclude_all(board);
	}while(fill_all(board));
	if(is_solveed(board))
		return true;
	else{
		Board copy(board);
		uint where = find_best_guess(board);
		for(uint i=0; i<9; ++i){
			if(board[where] & (1<<i)){
				copy[where] = (1<<i);

				try{
					if(solve(copy)){
						board.swap(copy);
						return true;
					}
				}catch(no_solution&){
				}
			}
		}
		return false;
	}
};

int main(){
	ifstream in("SUDO_IN.TXT");
	ofstream out("SUDO_OUT.TXT");
	try{
		Board board(9*9);
		readin(in, board);

		if(!solve(board))
			throw no_solution();

		writeout(out, board);
	}catch(no_solution&){
		out<<"Pas de solution";
	}
}
```