C++ Software Engineering

for engineers of other disciplines

Mini Project

Evaluation

"What not to do!"

Summer 2020

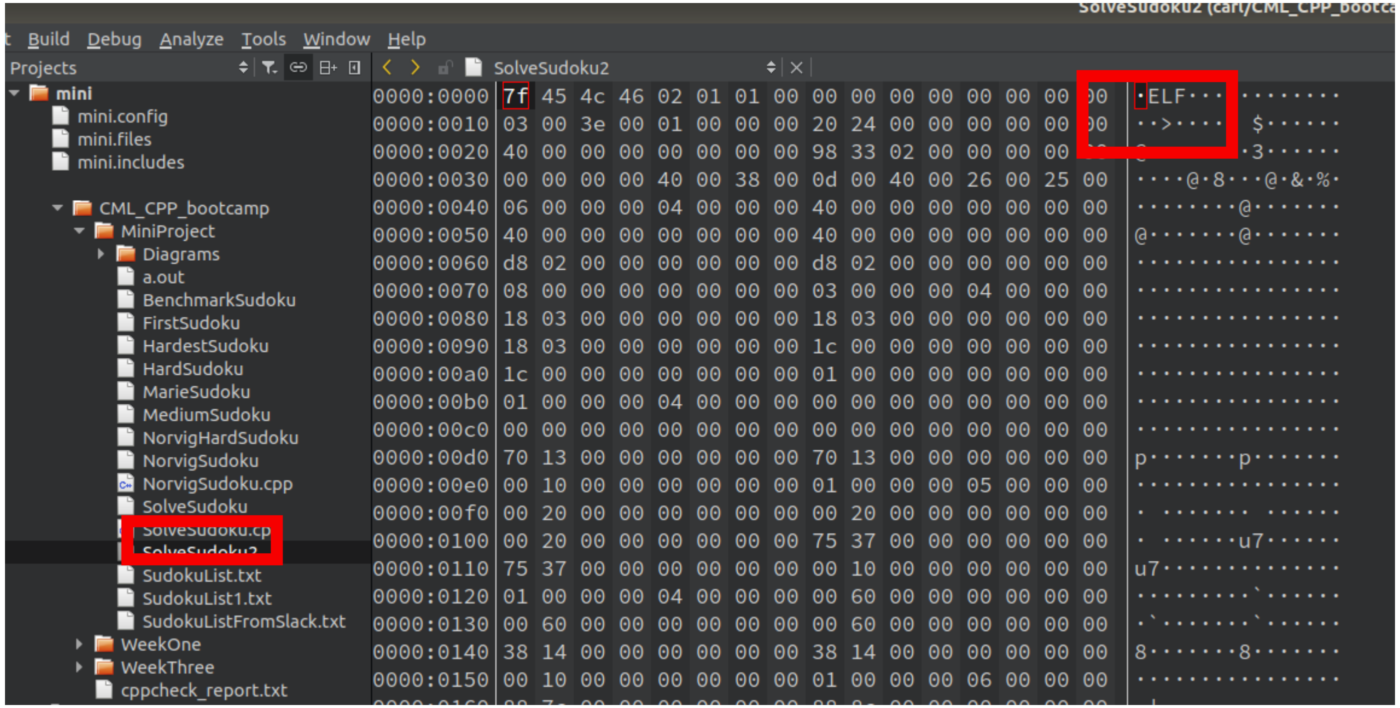
Gothenburg, Sweden

rashid.zamani@alten.se

What to push to git?

- Only document is uploaded to git! Git bundle for binary – pictures are also binaries! Text is document, asci!
- Rarely binaries are pushed to git – mostly through git LargeFileStorage (lfs) [1].

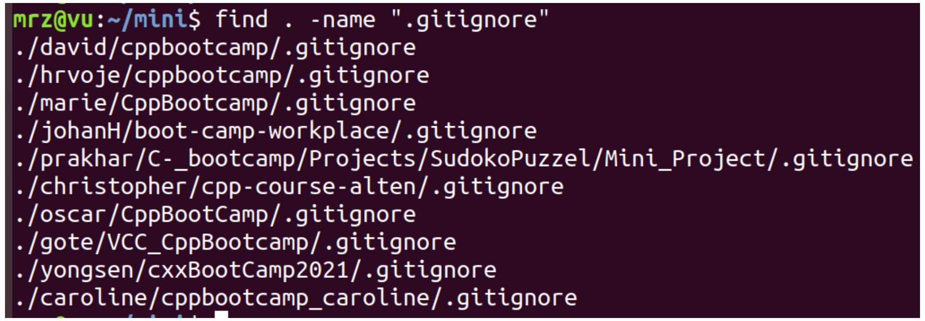
[1] Git Large File Storage | Git Large File Storage (LFS) replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.



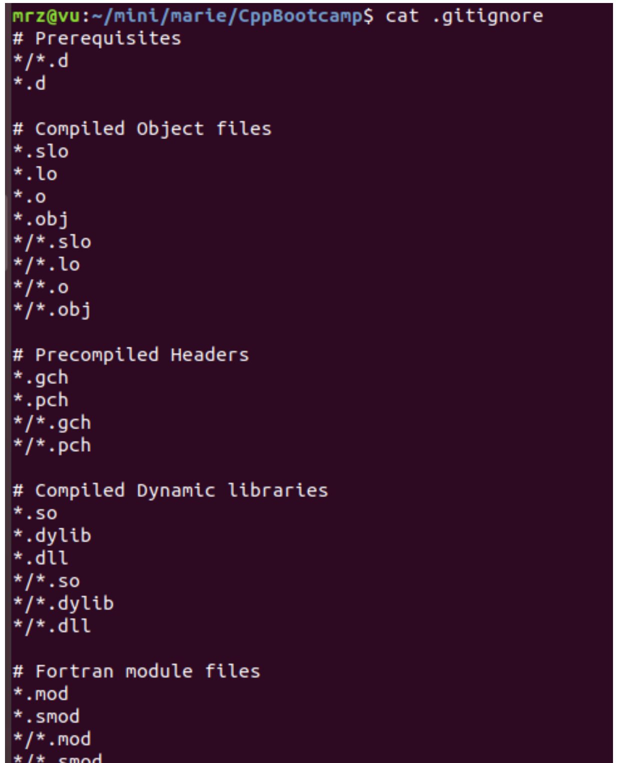
```
0000:0000 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  ·ELF············
0000:0010 03 00 3e 00 01 00 00 00 20 24 00 00 00 00 00 00  ··>····· $······
0000:0020 40 00 00 00 00 00 00 00 98 33 02 00 00 00 00 00  @········3······
0000:0030 00 00 00 00 40 00 38 00 0d 00 40 00 26 00 25 00  ····@·8···@·&·%·
0000:0040 06 00 00 00 04 00 00 00 40 00 00 00 00 00 00 00  ········@·······
0000:0050 40 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  @·······@·······
0000:0060 d8 02 00 00 00 00 00 00 d8 02 00 00 00 00 00 00  ················
0000:0070 08 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00  ················
0000:0080 18 03 00 00 00 00 00 00 18 03 00 00 00 00 00 00  ················
0000:0090 18 03 00 00 00 00 00 00 1c 00 00 00 00 00 00 00  ················
0000:00a0 1c 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ················
0000:00b0 01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00  ················
0000:00c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ················
0000:00d0 70 13 00 00 00 00 00 00 70 13 00 00 00 00 00 00  p·······p·······
0000:00e0 00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00  ················
0000:00f0 00 20 00 00 00 00 00 00 00 20 00 00 00 00 00 00  · ······· ······
0000:0100 00 20 00 00 00 00 00 00 75 37 00 00 00 00 00 00  · ······u7······
0000:0110 75 37 00 00 00 00 00 00 00 10 00 00 00 00 00 00  u7··············
0000:0120 01 00 00 00 04 00 00 00 00 60 00 00 00 00 00 00  ·········`······
0000:0130 00 60 00 00 00 00 00 00 00 60 00 00 00 00 00 00  ·`·······`······
0000:0140 38 14 00 00 00 00 00 00 38 14 00 00 00 00 00 00  8·······8·······
0000:0150 00 10 00 00 00 00 00 00 01 00 00 00 06 00 00 00  ················
```

Gitignore

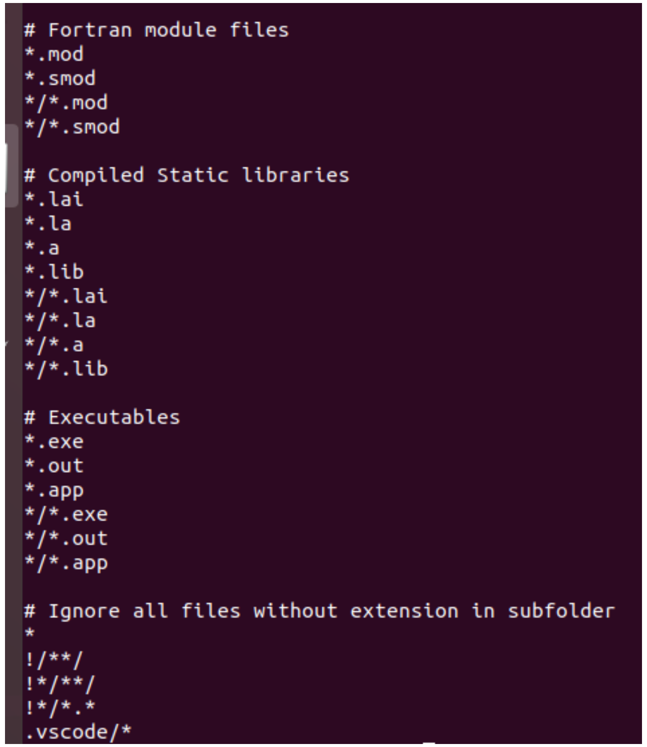
- Define patterns for file to not be tracked!
- Make it hidden by having a dot in the beginning.
- Push to git!
- git will ignore file patterns within “`.gitignore`”



```
mrz@vu:~/mini$ find . -name ".gitignore"
./david/cppbootcamp/.gitignore
./hrvoje/cppbootcamp/.gitignore
./marie/CppBootcamp/.gitignore
./johanH/boot-camp-workplace/.gitignore
./prakhar/C-_bootcamp/Projects/SudokoPuzzel/Mini_Project/.gitignore
./christopher/cpp-course-alten/.gitignore
./oscar/CppBootCamp/.gitignore
./gote/VCC_CppBootcamp/.gitignore
./yongsen/cxxBootCamp2021/.gitignore
./caroline/cppbootcamp_caroline/.gitignore
```



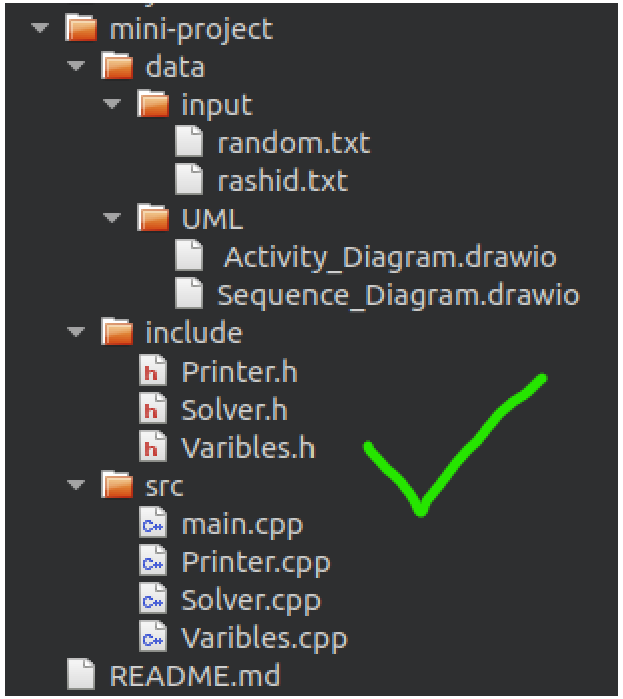
```
mrz@vu:~/mini/marie/CppBootcamp$ cat .gitignore
# Prerequisites
*/*.d
*.d

# Compiled Object files
*.slo
*.lo
*.o
*.obj
*/*.slo
*/*.lo
*/*.o
*/*.obj

# Precompiled Headers
*.gch
*.pch
*/*.gch
*/*.pch

# Compiled Dynamic libraries
*.so
*.dylib
*.dll
*/*.so
*/*.dylib
*/*.dll

# Fortran module files
*.mod
*.smod
*/*.mod
*/*.smod
```



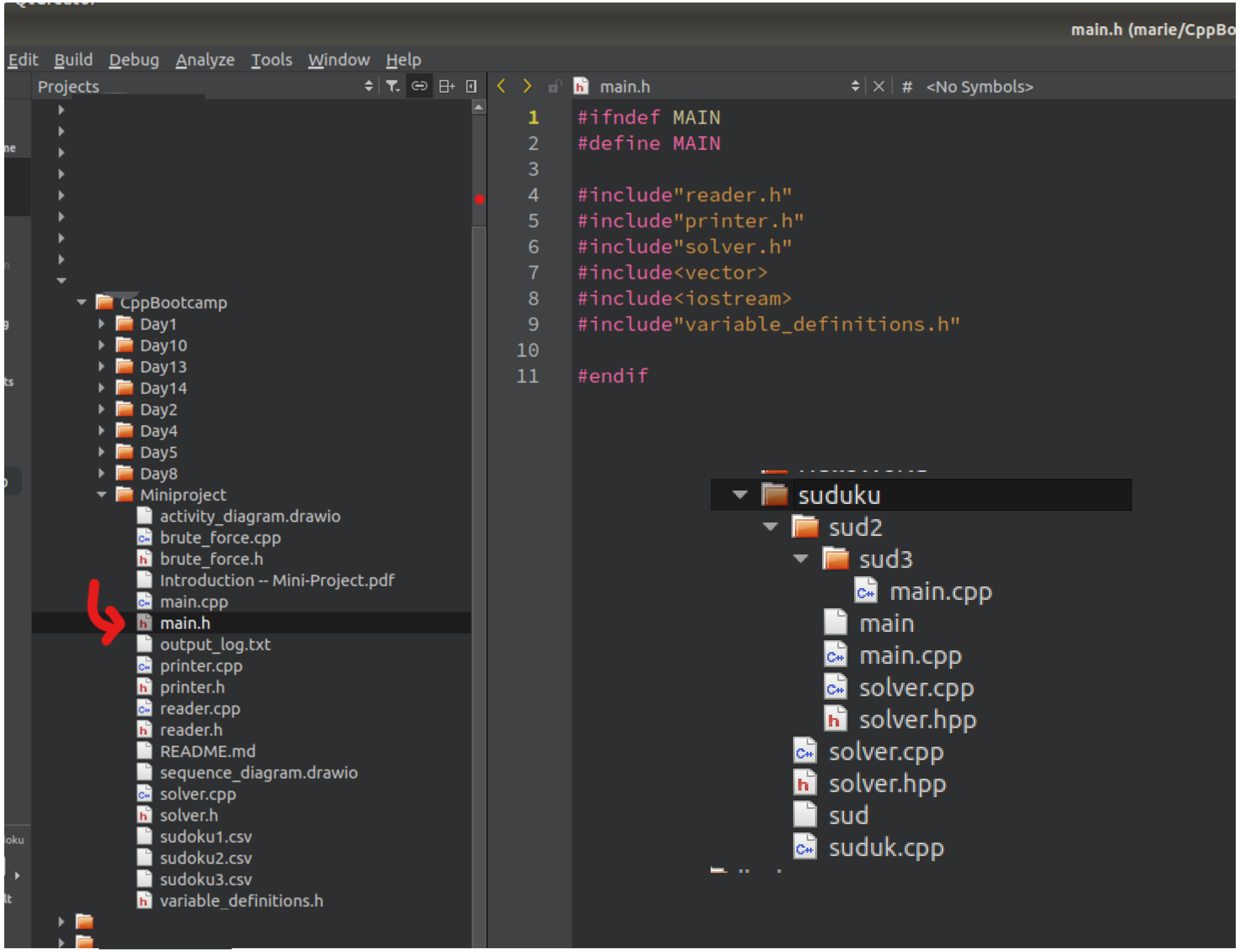
```
# Fortran module files
*.mod
*.smod
*/*.mod
*/*.smod

# Compiled Static libraries
*.lai
*.la
*.a
*.lib
*/*.lai
*/*.la
*/*.a
*/*.lib

# Executables
*.exe
*.out
*.app
*/*.exe
*/*.out
*/*.app

# Ignore all files without extension in subfolder
*
!/**/
!*/**/
!*/*.*
.vscode/*
```

Foldering structure

- Separate folder for source files.
- Separate folder for header files.
- Main outside in root?
- Makefile? Cmakelist.txt?



```
mini-project
  data
    input
      random.txt
      rashid.txt
    UML
      Activity_Diagram.drawio
      Sequence_Diagram.drawio
  include
    Printer.h
    Solver.h
    Varibles.h
  src
    main.cpp
    Printer.cpp
    Solver.cpp
    Varibles.cpp
  README.md
```



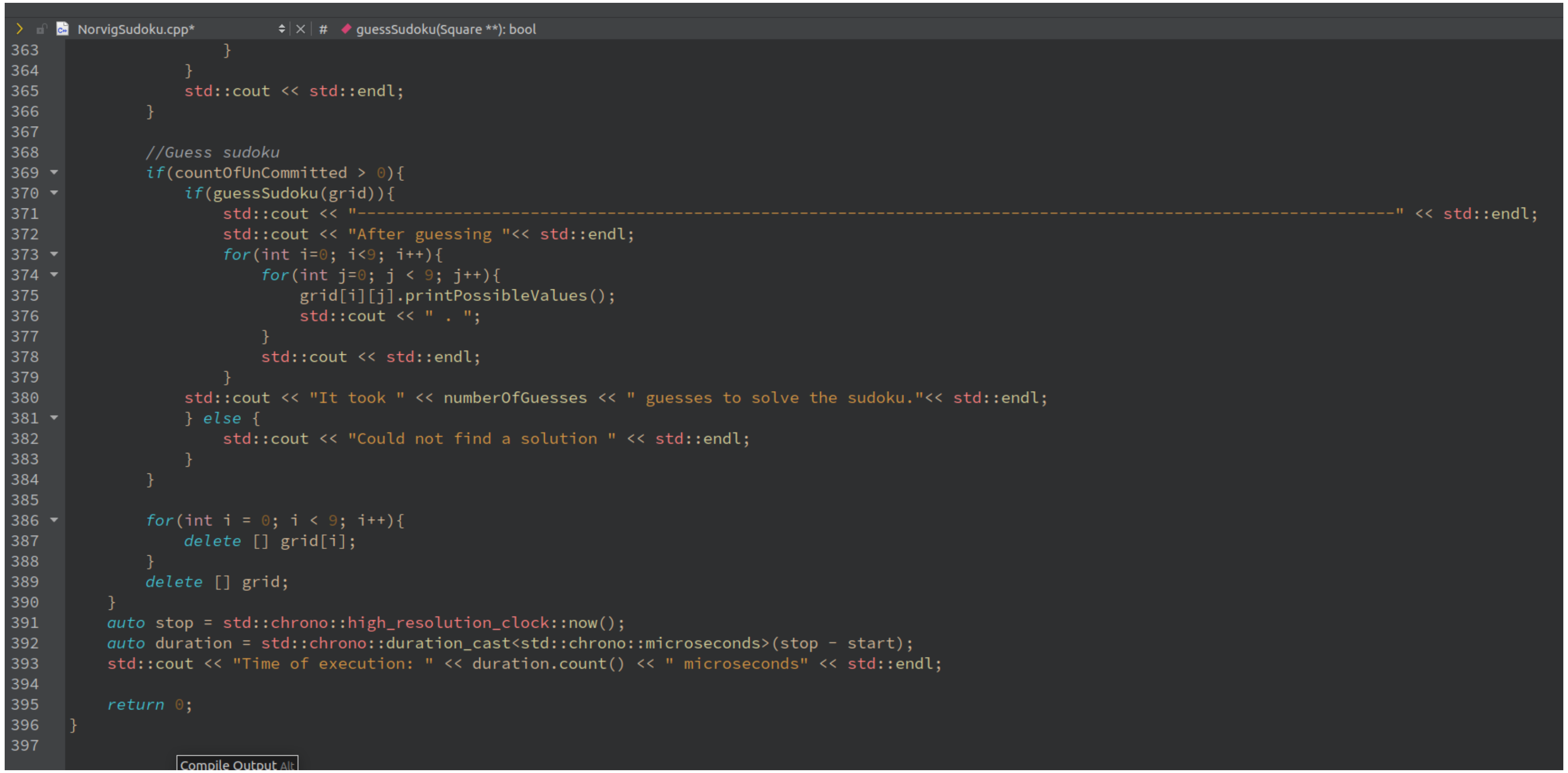
```
CppBootcamp
  Day1
  Day10
  Day13
  Day14
  Day2
  Day4
  Day5
  Day8
  Miniproject
    activity_diagram.drawio
    brute_force.cpp
    brute_force.h
    Introduction -- Mini-Project.pdf
    main.cpp
    main.h
    output_log.txt
    printer.cpp
    printer.h
    reader.cpp
    reader.h
    README.md
    sequence_diagram.drawio
    solver.cpp
    solver.h
    sudoku1.csv
    sudoku2.csv
    sudoku3.csv
    variable_definitions.h
```

```
#ifndef MAIN
#define MAIN

#include"reader.h"
#include"printer.h"
#include"solver.h"
#include<vector>
#include<iostream>
#include"variable_definitions.h"

#endif
```

```
suduku
  sud2
    sud3
      main.cpp
    main
    main.cpp
    solver.cpp
    solver.hpp
  solver.cpp
  solver.hpp
  sud
  suduk.cpp
```

main

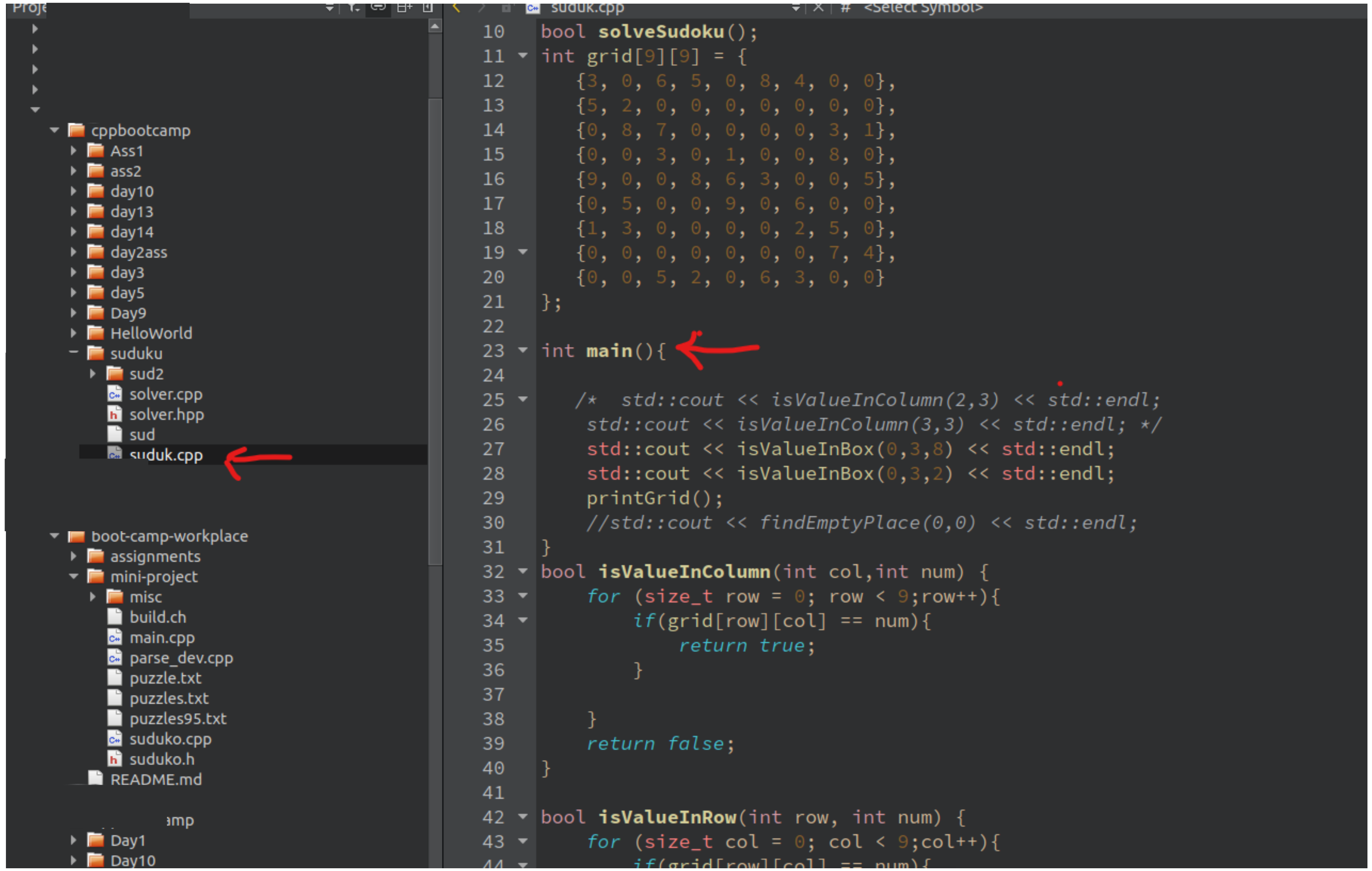
- Everything in one file!
- No header?
- Everything in **main**?
- What to put in **main**?



```cpp
                }
            }
            std::cout << std::endl;
        }

        //Guess sudoku
        if(countOfUnCommitted > 0){
            if(guessSudoku(grid)){
                std::cout << "------------------------------------------------------------------------------------------------------------" << std::endl;
                std::cout << "After guessing "<< std::endl;
                for(int i=0; i<9; i++){
                    for(int j=0; j < 9; j++){
                        grid[i][j].printPossibleValues();
                        std::cout << " . ";
                    }
                    std::cout << std::endl;
                }
            std::cout << "It took " << numberOfGuesses << " guesses to solve the sudoku."<< std::endl;
            } else {
                std::cout << "Could not find a solution " << std::endl;
            }
        }

        for(int i = 0; i < 9; i++){
            delete [] grid[i];
        }
        delete [] grid;
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);
    std::cout << "Time of execution: " << duration.count() << " microseconds" << std::endl;

    return 0;
}
```

main

- A .cpp file with the same name is recommended to have.

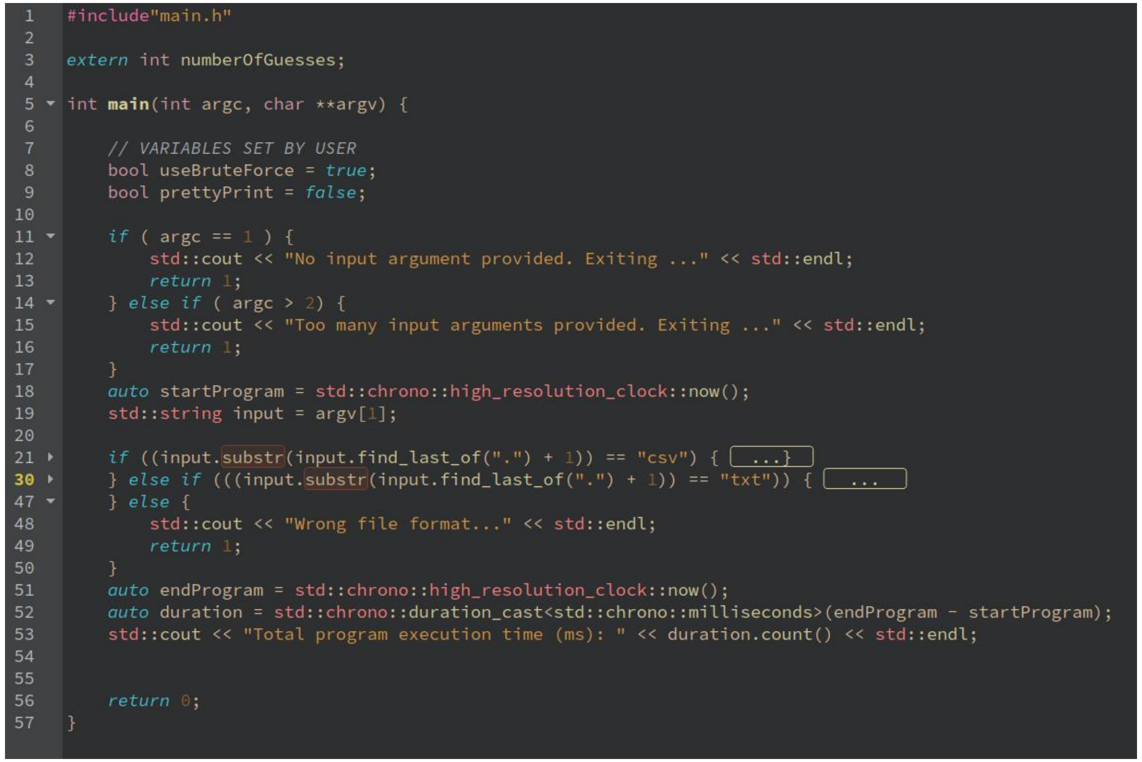


```
cppbootcamp
    Ass1
    ass2
    day10
    day13
    day14
    day2ass
    day3
    day5
    Day9
    HelloWorld
    suduku
        sud2
        solver.cpp
        solver.hpp
        sud
        suduk.cpp
boot-camp-workplace
    assignments
    mini-project
        misc
        build.ch
        main.cpp
        parse_dev.cpp
        puzzle.txt
        puzzles.txt
        puzzles95.txt
        suduko.cpp
        suduko.h
    README.md
amp
    Day1
    Day10
```

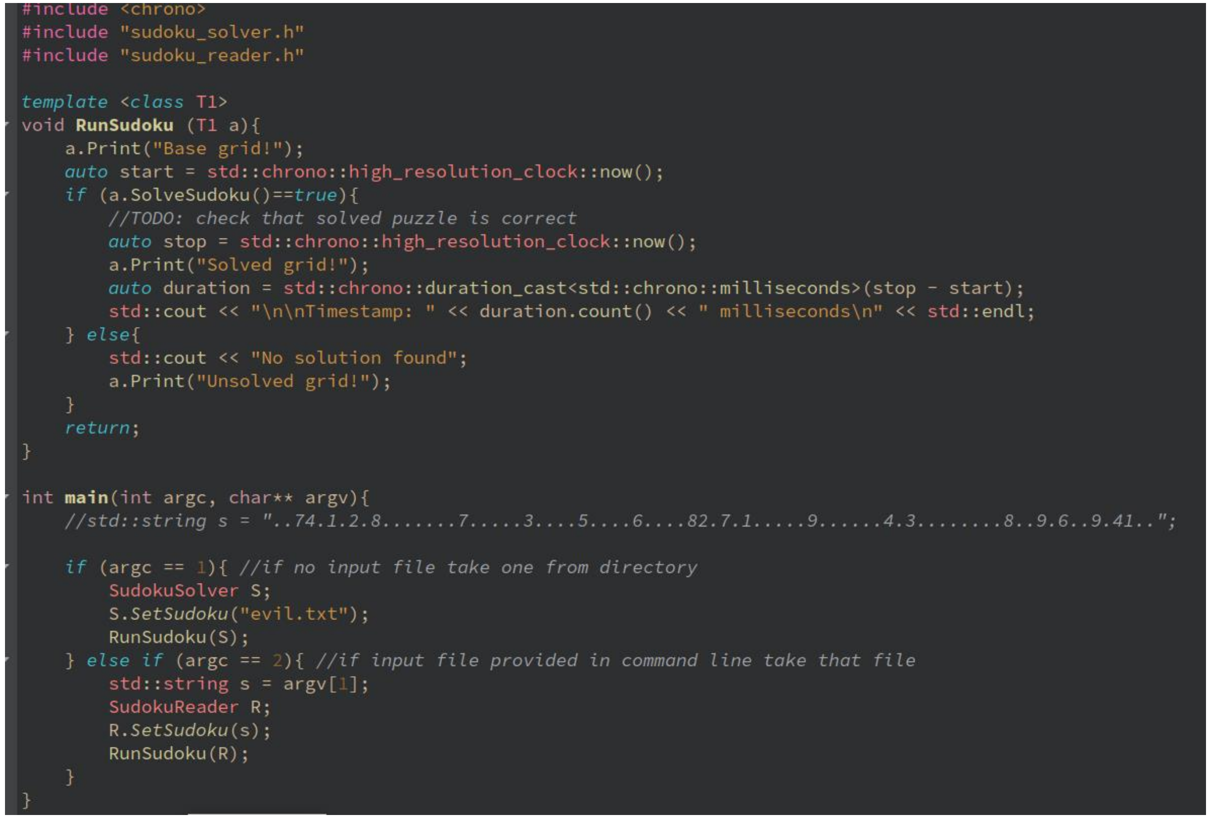
```cpp
bool solveSudoku();
int grid[9][9] = {
    {3, 0, 6, 5, 0, 8, 4, 0, 0},
    {5, 2, 0, 0, 0, 0, 0, 0, 0},
    {0, 8, 7, 0, 0, 0, 0, 3, 1},
    {0, 0, 3, 0, 1, 0, 0, 8, 0},
    {9, 0, 0, 8, 6, 3, 0, 0, 5},
    {0, 5, 0, 0, 9, 0, 6, 0, 0},
    {1, 3, 0, 0, 0, 0, 2, 5, 0},
    {0, 0, 0, 0, 0, 0, 0, 7, 4},
    {0, 0, 5, 2, 0, 6, 3, 0, 0}
};

int main(){

    /*  std::cout << isValueInColumn(2,3) << std::endl;
     std::cout << isValueInColumn(3,3) << std::endl; */
     std::cout << isValueInBox(0,3,8) << std::endl;
     std::cout << isValueInBox(0,3,2) << std::endl;
     printGrid();
     //std::cout << findEmptyPlace(0,0) << std::endl;
}
bool isValueInColumn(int col,int num) {
    for (size_t row = 0; row < 9;row++){
        if(grid[row][col] == num){
            return true;
        }

    }
    return false;
}

bool isValueInRow(int row, int num) {
    for (size_t col = 0; col < 9;col++){
        if(grid[row][col] == num){
```

main

- Two nice main – template for generic parts!
- Why not have logic there? How do you test?



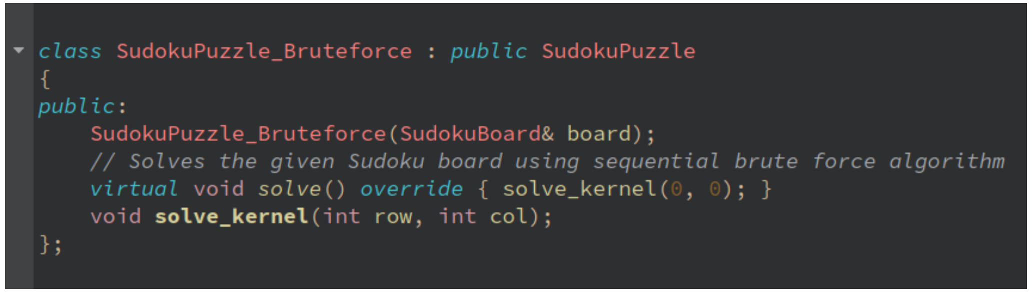
```cpp
#include"main.h"

extern int numberOfGuesses;

int main(int argc, char **argv) {

    // VARIABLES SET BY USER
    bool useBruteForce = true;
    bool prettyPrint = false;

    if ( argc == 1 ) {
        std::cout << "No input argument provided. Exiting ..." << std::endl;
        return 1;
    } else if ( argc > 2) {
        std::cout << "Too many input arguments provided. Exiting ..." << std::endl;
        return 1;
    }
    auto startProgram = std::chrono::high_resolution_clock::now();
    std::string input = argv[1];

    if ((input.substr(input.find_last_of(".") + 1)) == "csv") { ...}
    } else if (((input.substr(input.find_last_of(".") + 1)) == "txt")) { ...
    } else {
        std::cout << "Wrong file format..." << std::endl;
        return 1;
    }
    auto endProgram = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endProgram - startProgram);
    std::cout << "Total program execution time (ms): " << duration.count() << std::endl;


    return 0;
}
```



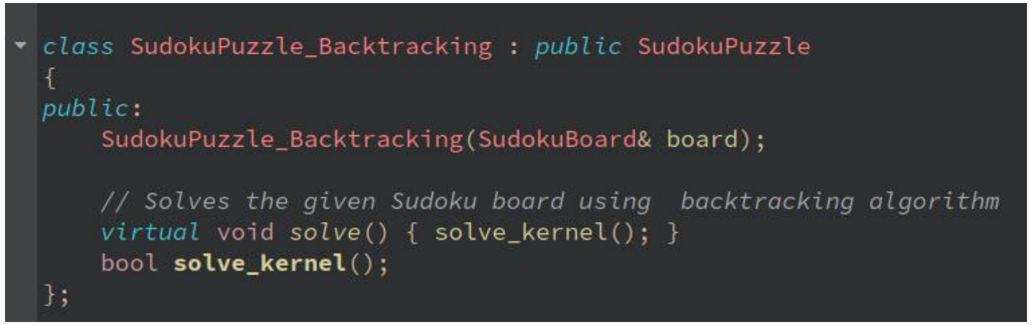
```cpp
#include <chrono>
#include "sudoku_solver.h"
#include "sudoku_reader.h"

template <class T1>
void RunSudoku (T1 a){
    a.Print("Base grid!");
    auto start = std::chrono::high_resolution_clock::now();
    if (a.SolveSudoku()==true){
        //TODO: check that solved puzzle is correct
        auto stop = std::chrono::high_resolution_clock::now();
        a.Print("Solved grid!");
        auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(stop - start);
        std::cout << "\n\nTimestamp: " << duration.count() << " milliseconds\n" << std::endl;
    } else{
        std::cout << "No solution found";
        a.Print("Unsolved grid!");
    }
    return;
}

int main(int argc, char** argv){
    //std::string s = "..74.1.2.8.......7.....3....5....6....82.7.1.....9......4.3........8..9.6..9.41..";

    if (argc == 1){ //if no input file take one from directory
        SudokuSolver S;
        S.SetSudoku("evil.txt");
        RunSudoku(S);
    } else if (argc == 2){ //if input file provided in command line take that file
        std::string s = argv[1];
        SudokuReader R;
        R.SetSudoku(s);
        RunSudoku(R);
    }
}
```

Base class deduction

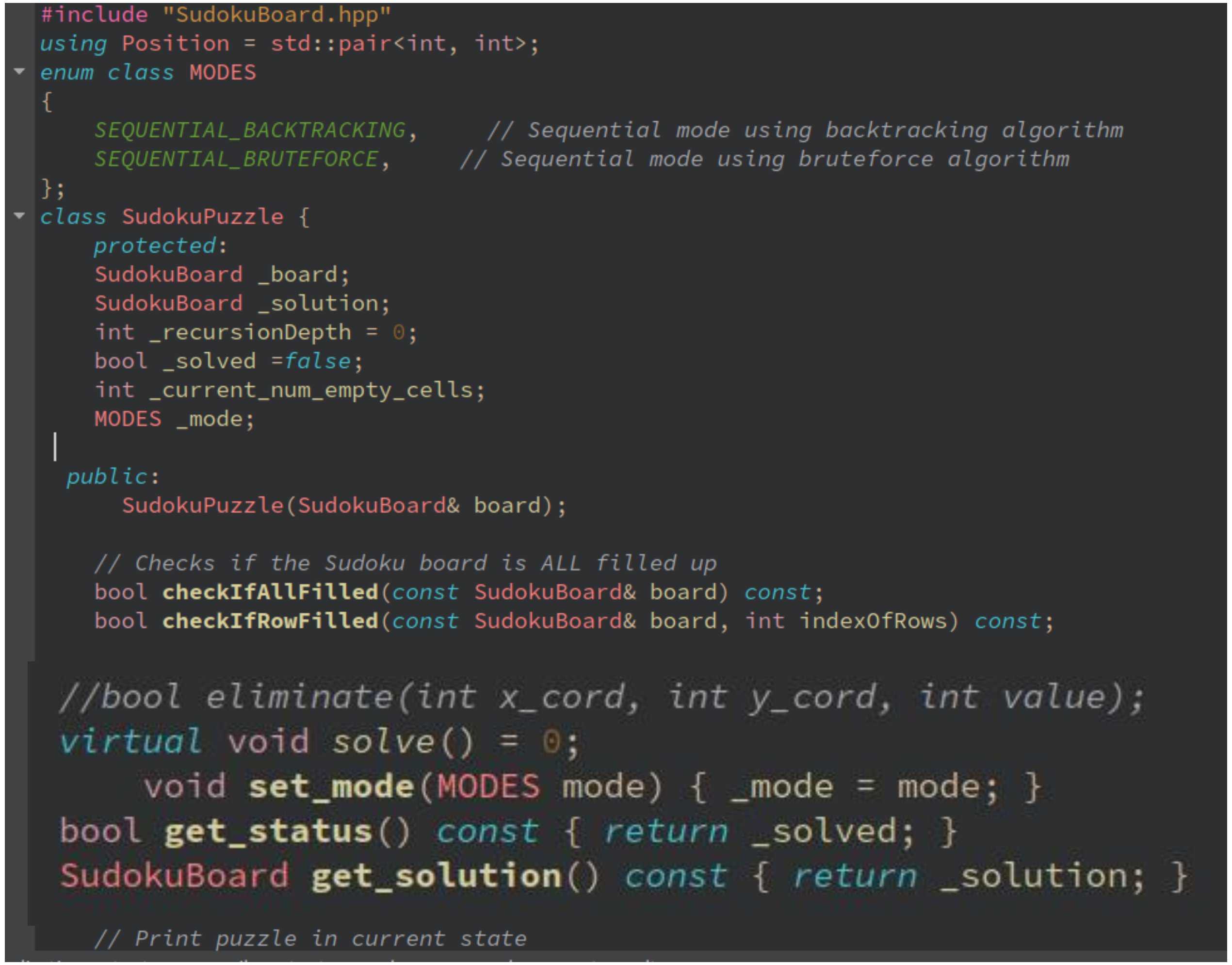
- Different algorithms could be invoked the same!



```cpp
class SudokuPuzzle_Bruteforce : public SudokuPuzzle
{
public:
    SudokuPuzzle_Bruteforce(SudokuBoard& board);
    // Solves the given Sudoku board using sequential brute force algorithm
    virtual void solve() override { solve_kernel(0, 0); }
    void solve_kernel(int row, int col);
};
```



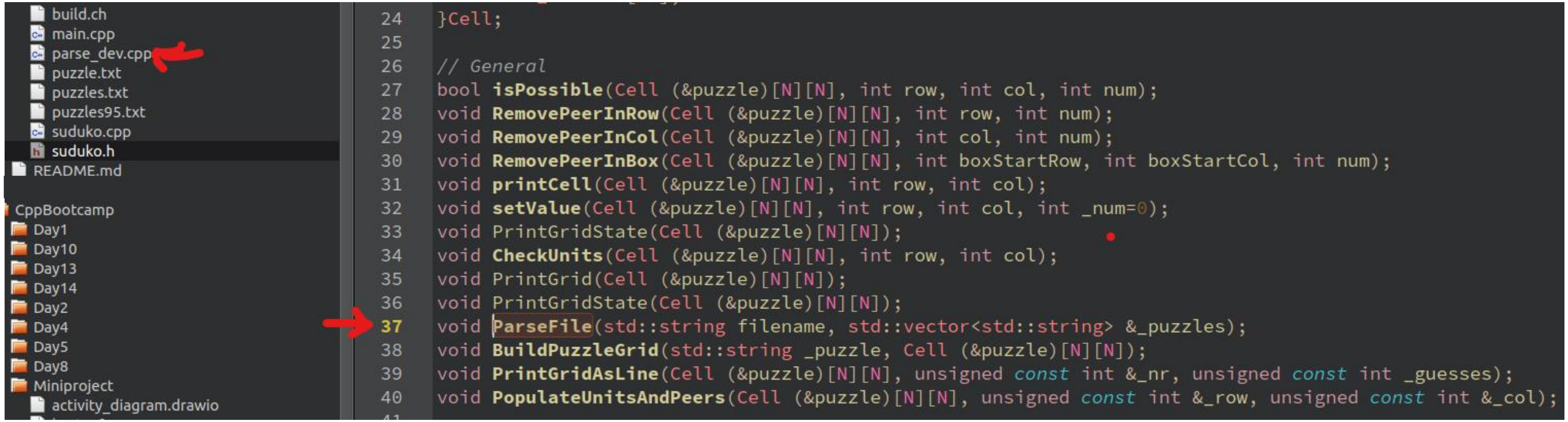
```cpp
class SudokuPuzzle_Backtracking : public SudokuPuzzle
{
public:
    SudokuPuzzle_Backtracking(SudokuBoard& board);

    // Solves the given Sudoku board using  backtracking algorithm
    virtual void solve() { solve_kernel(); }
    bool solve_kernel();
};
```



```cpp
#include "SudokuBoard.hpp"
using Position = std::pair<int, int>;
enum class MODES
{
    SEQUENTIAL_BACKTRACKING,     // Sequential mode using backtracking algorithm
    SEQUENTIAL_BRUTEFORCE,     // Sequential mode using bruteforce algorithm
};
class SudokuPuzzle {
    protected:
    SudokuBoard _board;
    SudokuBoard _solution;
    int _recursionDepth = 0;
    bool _solved =false;
    int _current_num_empty_cells;
    MODES _mode;

  public:
      SudokuPuzzle(SudokuBoard& board);

    // Checks if the Sudoku board is ALL filled up
    bool checkIfAllFilled(const SudokuBoard& board) const;
    bool checkIfRowFilled(const SudokuBoard& board, int indexOfRows) const;

//bool eliminate(int x_cord, int y_cord, int value);
virtual void solve() = 0;
    void set_mode(MODES mode) { _mode = mode; }
bool get_status() const { return _solved; }
SudokuBoard get_solution() const { return _solution; }

    // Print puzzle in current state
```

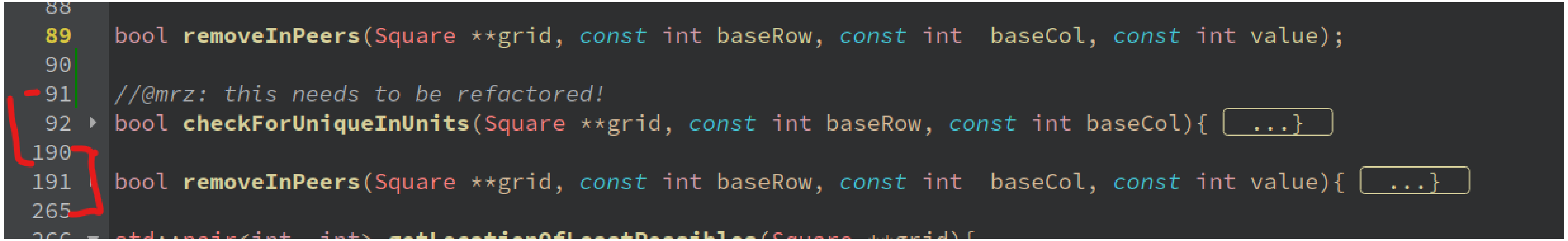
Header

- It is recommended to have a .cpp for any header if it has a function.
- Lazy people might have two header and one cpp if it is not too many function for faster linkage they argue, yet never the other way!!!



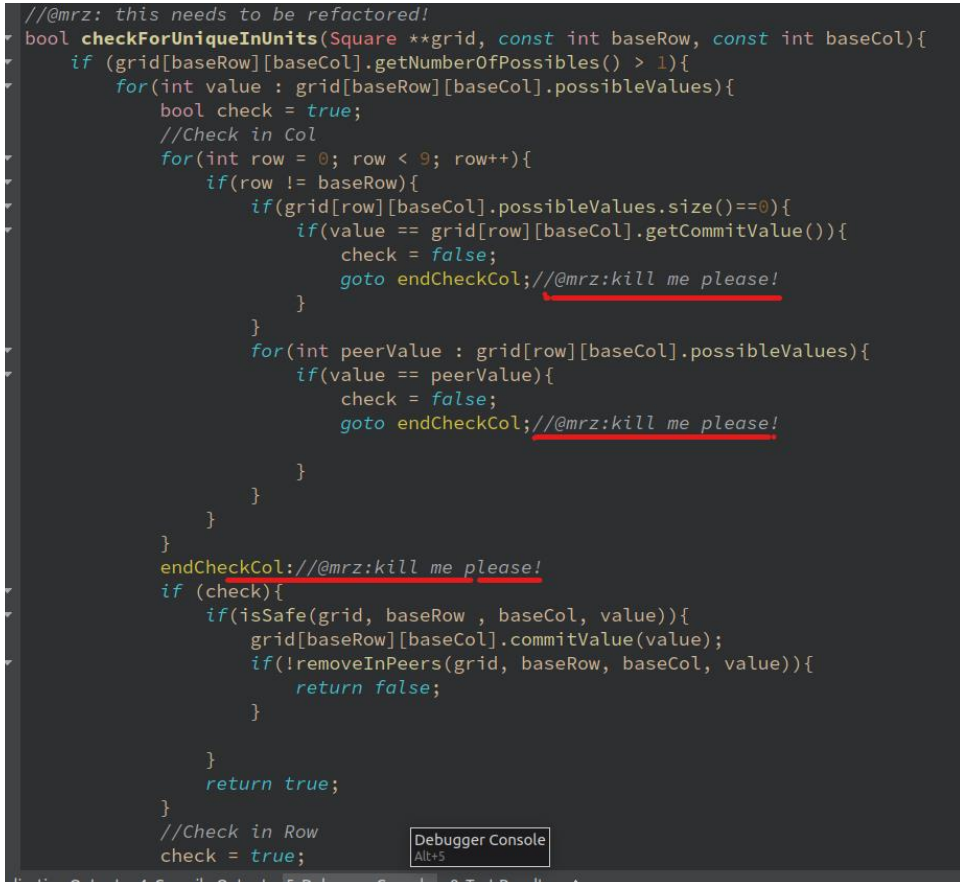
```
}Cell;

// General
bool isPossible(Cell (&puzzle)[N][N], int row, int col, int num);
void RemovePeerInRow(Cell (&puzzle)[N][N], int row, int num);
void RemovePeerInCol(Cell (&puzzle)[N][N], int col, int num);
void RemovePeerInBox(Cell (&puzzle)[N][N], int boxStartRow, int boxStartCol, int num);
void printCell(Cell (&puzzle)[N][N], int row, int col);
void setValue(Cell (&puzzle)[N][N], int row, int col, int _num=0);
void PrintGridState(Cell (&puzzle)[N][N]);
void CheckUnits(Cell (&puzzle)[N][N], int row, int col);
void PrintGrid(Cell (&puzzle)[N][N]);
void PrintGridState(Cell (&puzzle)[N][N]);
void ParseFile(std::string filename, std::vector<std::string> &_puzzles);
void BuildPuzzleGrid(std::string _puzzle, Cell (&puzzle)[N][N]);
void PrintGridAsLine(Cell (&puzzle)[N][N], unsigned const int &_nr, unsigned const int _guesses);
void PopulateUnitsAndPeers(Cell (&puzzle)[N][N], unsigned const int &_row, unsigned const int &_col);
```

Long functions are bad!



```
bool removeInPeers(Square **grid, const int baseRow, const int  baseCol, const int value);

//@mrz: this needs to be refactored!
bool checkForUniqueInUnits(Square **grid, const int baseRow, const int baseCol){ ...}
bool removeInPeers(Square **grid, const int baseRow, const int  baseCol, const int value){ ...}
std::pair<int, int> getLocationOfLeastPossibles(Square **grid){
```

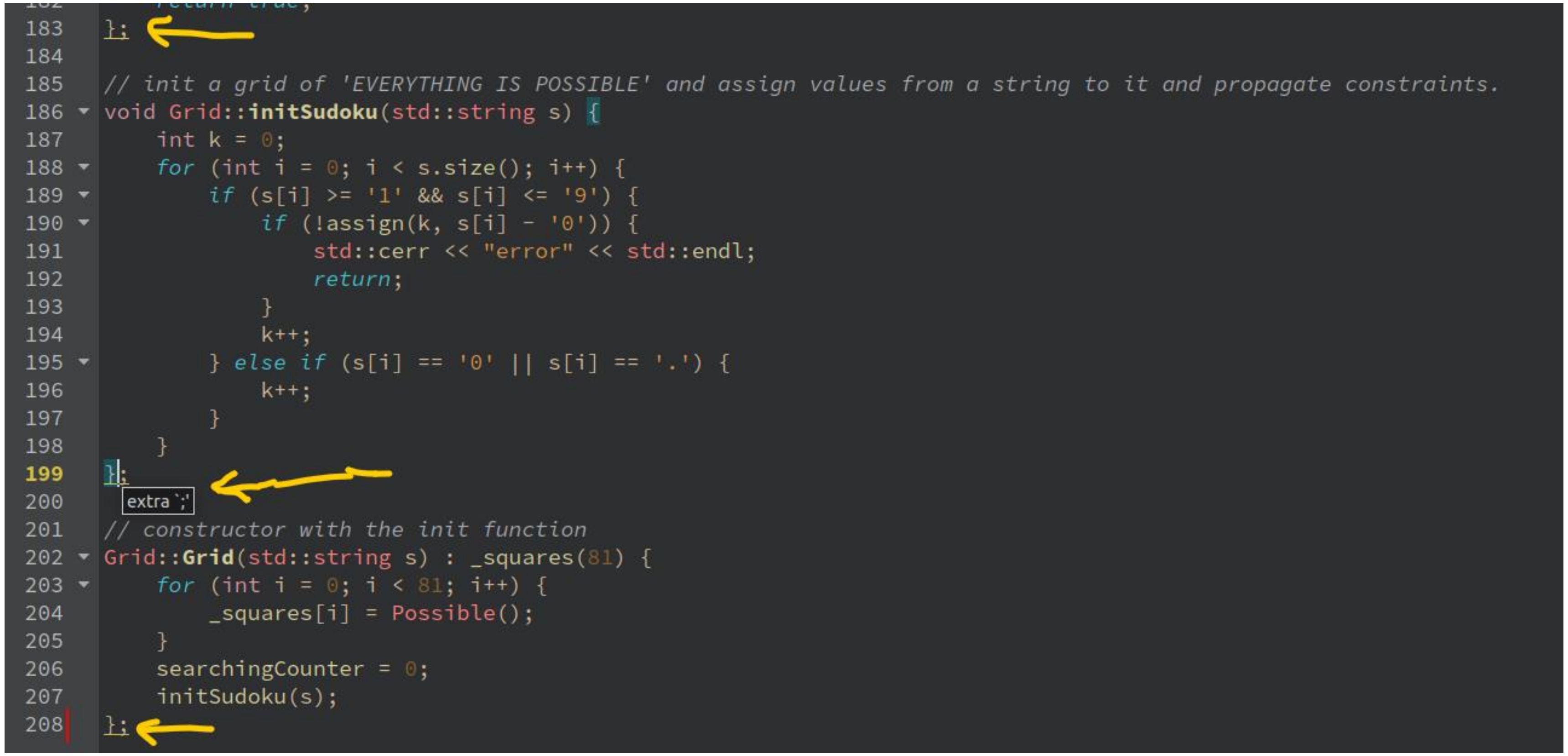
Goto



```
//@mrz: this needs to be refactored!
bool checkForUniqueInUnits(Square **grid, const int baseRow, const int baseCol){
    if (grid[baseRow][baseCol].getNumberOfPossibles() > 1){
        for(int value : grid[baseRow][baseCol].possibleValues){
            bool check = true;
            //Check in Col
            for(int row = 0; row < 9; row++){
                if(row != baseRow){
                    if(grid[row][baseCol].possibleValues.size()==0){
                        if(value == grid[row][baseCol].getCommitValue()){
                            check = false;
                            goto endCheckCol;//@mrz:kill me please!
                        }
                    }
                    for(int peerValue : grid[row][baseCol].possibleValues){
                        if(value == peerValue){
                            check = false;
                            goto endCheckCol;//@mrz:kill me please!

                        }
                    }
                }
            }
            endCheckCol://@mrz:kill me please!
            if (check){
                if(isSafe(grid, baseRow , baseCol, value)){
                    grid[baseRow][baseCol].commitValue(value);
                    if(!removeInPeers(grid, baseRow, baseCol, value)){
                        return false;
                    }

                }
                return true;
            }
            //Check in Row
            check = true;
```

;

- Semicolons are only needed at the end of a statement, like a struct, or class not a function!

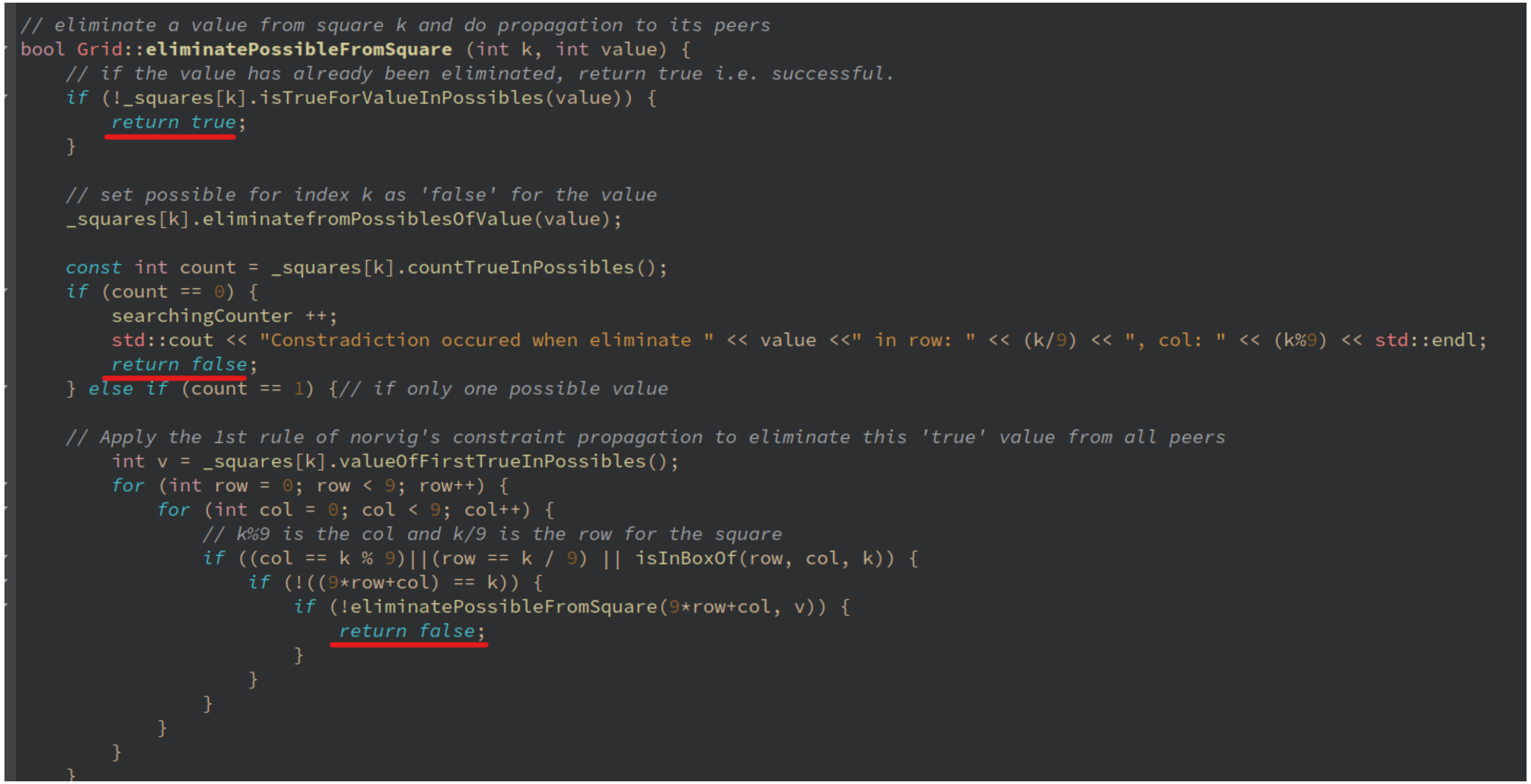


```
    return true;
};

// init a grid of 'EVERYTHING IS POSSIBLE' and assign values from a string to it and propagate constraints.
void Grid::initSudoku(std::string s) {
    int k = 0;
    for (int i = 0; i < s.size(); i++) {
        if (s[i] >= '1' && s[i] <= '9') {
            if (!assign(k, s[i] - '0')) {
                std::cerr << "error" << std::endl;
                return;
            }
            k++;
        } else if (s[i] == '0' || s[i] == '.') {
            k++;
        }
    }
};

// constructor with the init function
Grid::Grid(std::string s) : _squares(81) {
    for (int i = 0; i < 81; i++) {
        _squares[i] = Possible();
    }
    searchingCounter = 0;
    initSudoku(s);
};
```

extra ';'

return

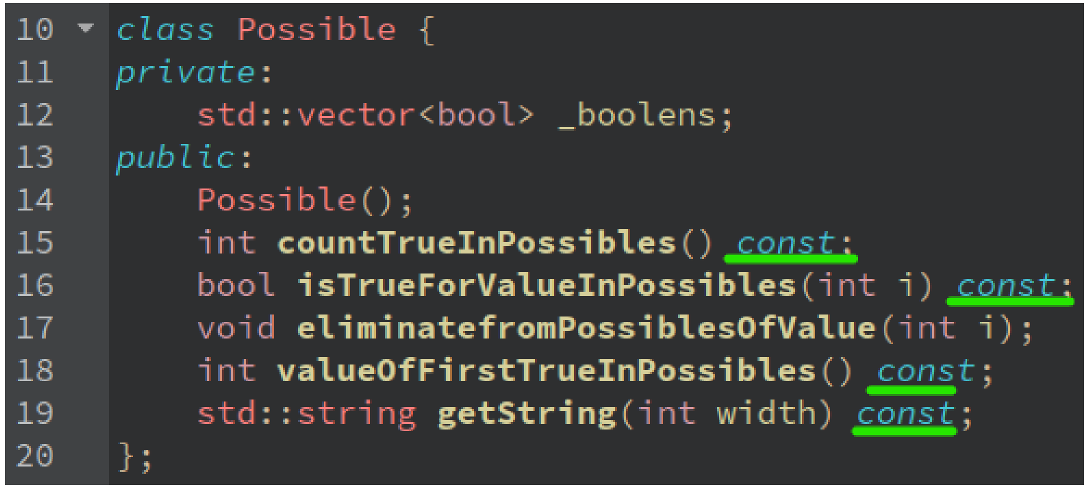
- Try not to return in the middle of a function.



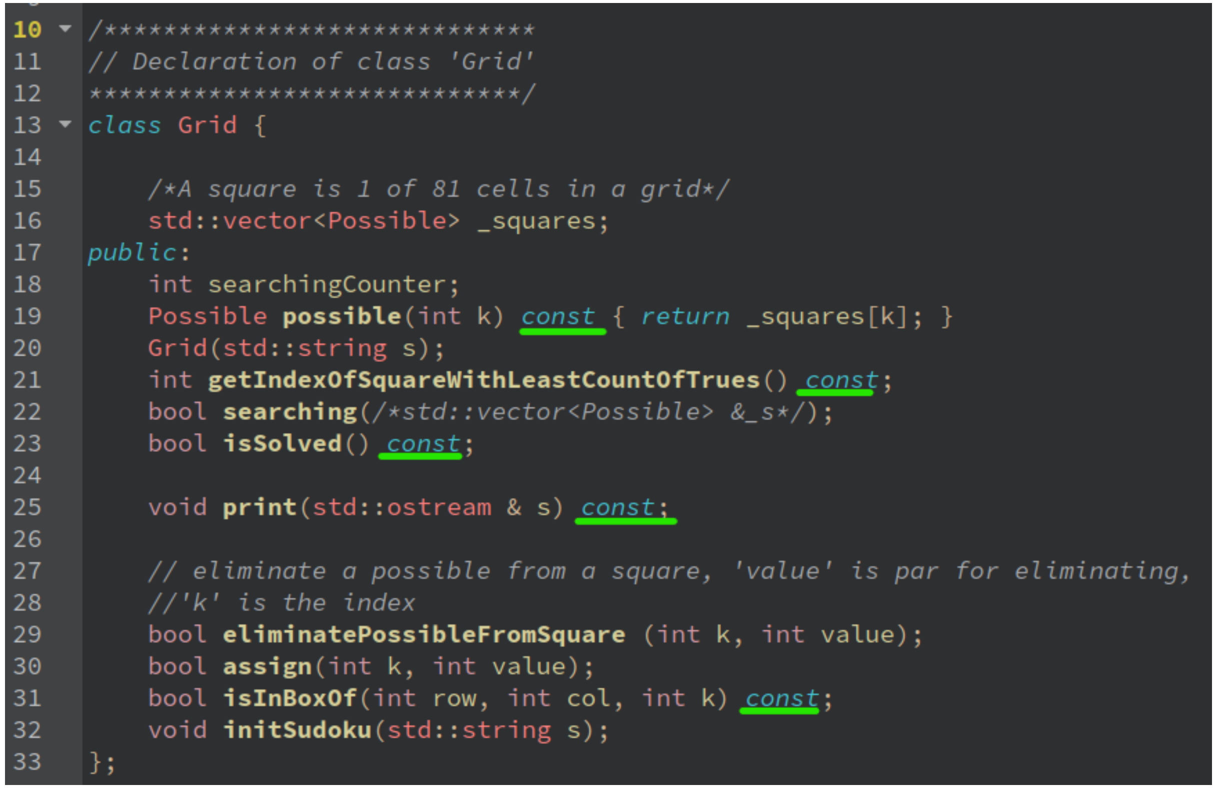
```cpp
// eliminate a value from square k and do propagation to its peers
bool Grid::eliminatePossibleFromSquare (int k, int value) {
    // if the value has already been eliminated, return true i.e. successful.
    if (!_squares[k].isTrueForValueInPossibles(value)) {
        return true;
    }

    // set possible for index k as 'false' for the value
    _squares[k].eliminatefromPossiblesOfValue(value);

    const int count = _squares[k].countTrueInPossibles();
    if (count == 0) {
        searchingCounter ++;
        std::cout << "Constradiction occured when eliminate " << value <<" in row: " << (k/9) << ", col: " << (k%9) << std::endl;
        return false;
    } else if (count == 1) {// if only one possible value

    // Apply the 1st rule of norvig's constraint propagation to eliminate this 'true' value from all peers
        int v = _squares[k].valueOfFirstTrueInPossibles();
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                // k%9 is the col and k/9 is the row for the square
                if ((col == k % 9)||(row == k / 9) || isInBoxOf(row, col, k)) {
                    if (!((9*row+col) == k)) {
                        if (!eliminatePossibleFromSquare(9*row+col, v)) {
                            return false;
                        }
                    }
                }
            }
        }
    }
```

const

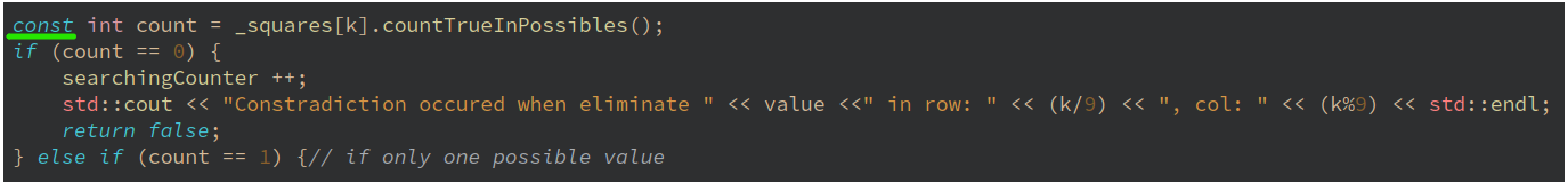
- Use the keyword, as much as they make sense!



```cpp
class Possible {
private:
    std::vector<bool> _boolens;
public:
    Possible();
    int countTrueInPossibles() const;
    bool isTrueForValueInPossibles(int i) const;
    void eliminatefromPossiblesOfValue(int i);
    int valueOfFirstTrueInPossibles() const;
    std::string getString(int width) const;
};
```

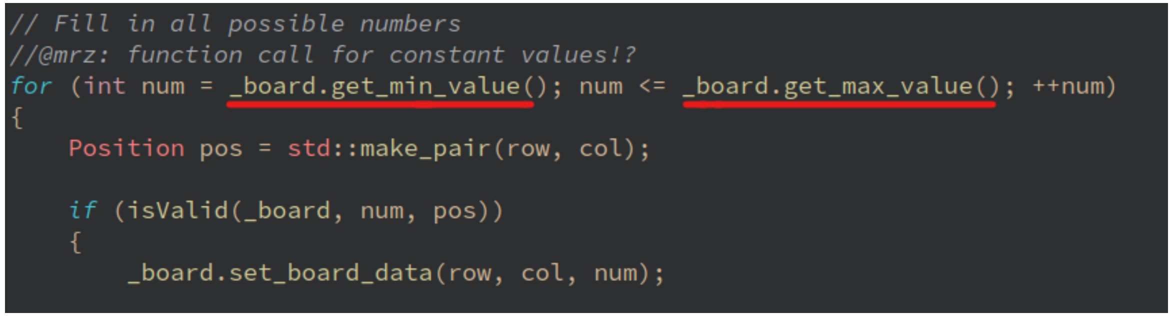


```cpp
/***************************
// Declaration of class 'Grid'
***************************/
class Grid {

    /*A square is 1 of 81 cells in a grid*/
    std::vector<Possible> _squares;
public:
    int searchingCounter;
    Possible possible(int k) const { return _squares[k]; }
    Grid(std::string s);
    int getIndexOfSquareWithLeastCountOfTrues() const;
    bool searching(/*std::vector<Possible> &_s*/);
    bool isSolved() const;

    void print(std::ostream & s) const;

    // eliminate a possible from a square, 'value' is par for eliminating,
    //'k' is the index
    bool eliminatePossibleFromSquare (int k, int value);
    bool assign(int k, int value);
    bool isInBoxOf(int row, int col, int k) const;
    void initSudoku(std::string s);
};
```

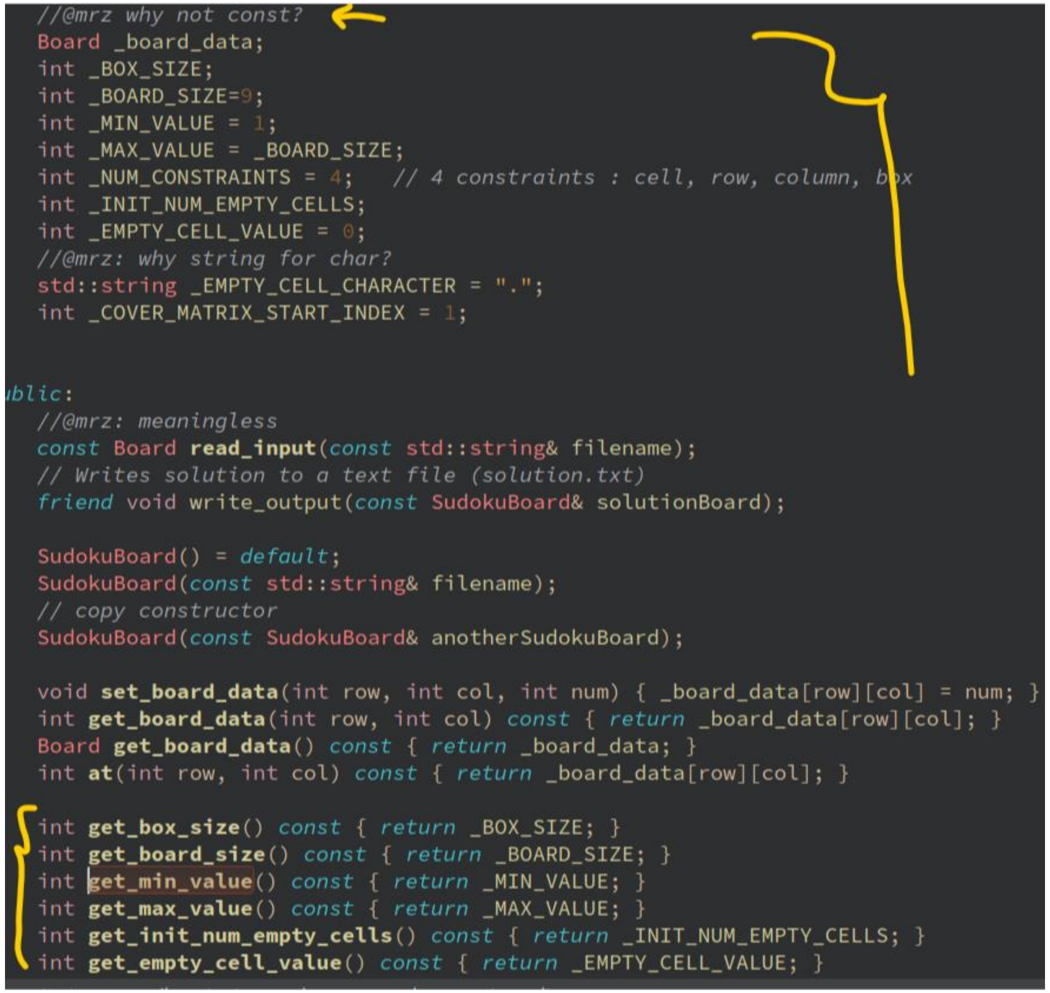


```cpp
const int count = _squares[k].countTrueInPossibles();
if (count == 0) {
    searchingCounter ++;
    std::cout << "Constradiction occured when eliminate " << value <<" in row: " << (k/9) << ", col: " << (k%9) << std::endl;
    return false;
} else if (count == 1) {// if only one possible value
```

const



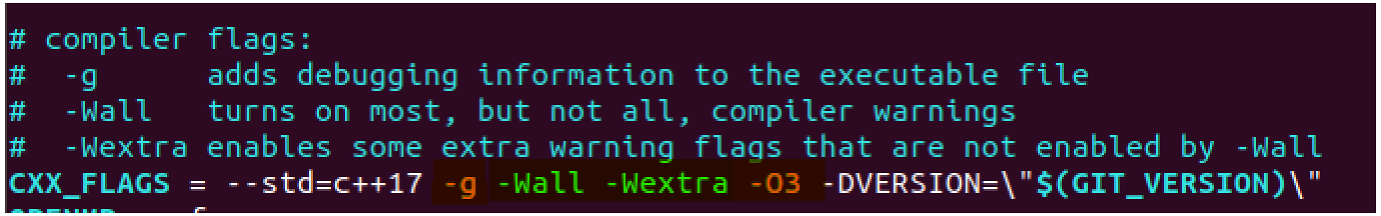
```
// Fill in all possible numbers
//@mrz: function call for constant values!?
for (int num = _board.get_min_value(); num <= _board.get_max_value(); ++num)
{
    Position pos = std::make_pair(row, col);

    if (isValid(_board, num, pos))
    {
        _board.set_board_data(row, col, num);
```



```
   //@mrz why not const?
   Board _board_data;
   int _BOX_SIZE;
   int _BOARD_SIZE=9;
   int _MIN_VALUE = 1;
   int _MAX_VALUE = _BOARD_SIZE;
   int _NUM_CONSTRAINTS = 4;   // 4 constraints : cell, row, column, box
   int _INIT_NUM_EMPTY_CELLS;
   int _EMPTY_CELL_VALUE = 0;
   //@mrz: why string for char?
   std::string _EMPTY_CELL_CHARACTER = ".";
   int _COVER_MATRIX_START_INDEX = 1;


ublic:
   //@mrz: meaningless
   const Board read_input(const std::string& filename);
   // Writes solution to a text file (solution.txt)
   friend void write_output(const SudokuBoard& solutionBoard);

   SudokuBoard() = default;
   SudokuBoard(const std::string& filename);
   // copy constructor
   SudokuBoard(const SudokuBoard& anotherSudokuBoard);

   void set_board_data(int row, int col, int num) { _board_data[row][col] = num; }
   int get_board_data(int row, int col) const { return _board_data[row][col]; }
   Board get_board_data() const { return _board_data; }
   int at(int row, int col) const { return _board_data[row][col]; }

   int get_box_size() const { return _BOX_SIZE; }
   int get_board_size() const { return _BOARD_SIZE; }
   int get_min_value() const { return _MIN_VALUE; }
   int get_max_value() const { return _MAX_VALUE; }
   int get_init_num_empty_cells() const { return _INIT_NUM_EMPTY_CELLS; }
   int get_empty_cell_value() const { return _EMPTY_CELL_VALUE; }
```

Overcomplexitinessfull!

- Use things which makes sense!



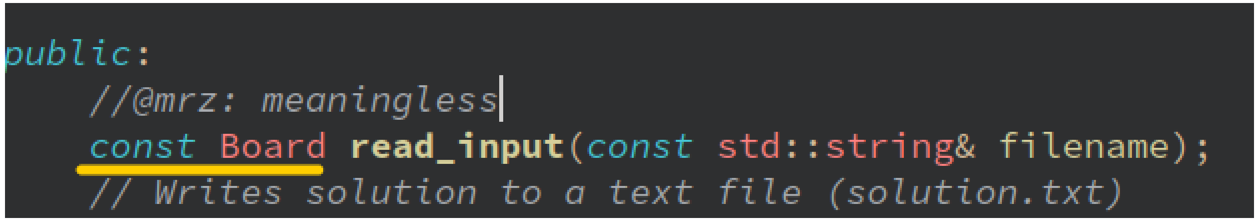
```
# compiler flags:
#  -g      adds debugging information to the executable file
#  -Wall   turns on most, but not all, compiler warnings
#  -Wextra enables some extra warning flags that are not enabled by -Wall
CXX_FLAGS = --std=c++17 -g -Wall -Wextra -O3 -DVERSION=\"$(GIT_VERSION)\"
```

Overcomplexitinessfull!

- The squares(state) is tiedd to grid, and the logic – grid is dead, what if another logic needs to be applied to your grid?! GOD object!

```
//: C08:ConstReturnValues.cpp
// Constant return by value
// Result cannot be used as an lvalue

class X {
  int i;
public:
  X(int ii = 0);
  void modify();
};

X::X(int ii) { i = ii; }

void X::modify() { i++; }

X f5() {
  return X();
}

const X f6() {
  return X();
}

void f7(X& x) { // Pass by non-const reference
  x.modify();
}

int main() {
  f5() = X(1); // OK -- non-const return value
  f5().modify(); // OK
//!  f7(f5()); // Causes warning or error
// Causes compile-time errors:
//!  f7(f5());
//!  f6() = X(1);
//!  f6().modify();
//!  f7(f6());
} ///:~
```

```
//: C08:Constval.cpp
// Returning consts by value
// has no meaning for built-in types

int f3() { return 1; }
const int f4() { return 1; }

int main() {
  const int j = f3(); // Works fine
  int k = f4(); // But this works fine too!
} ///:~
```

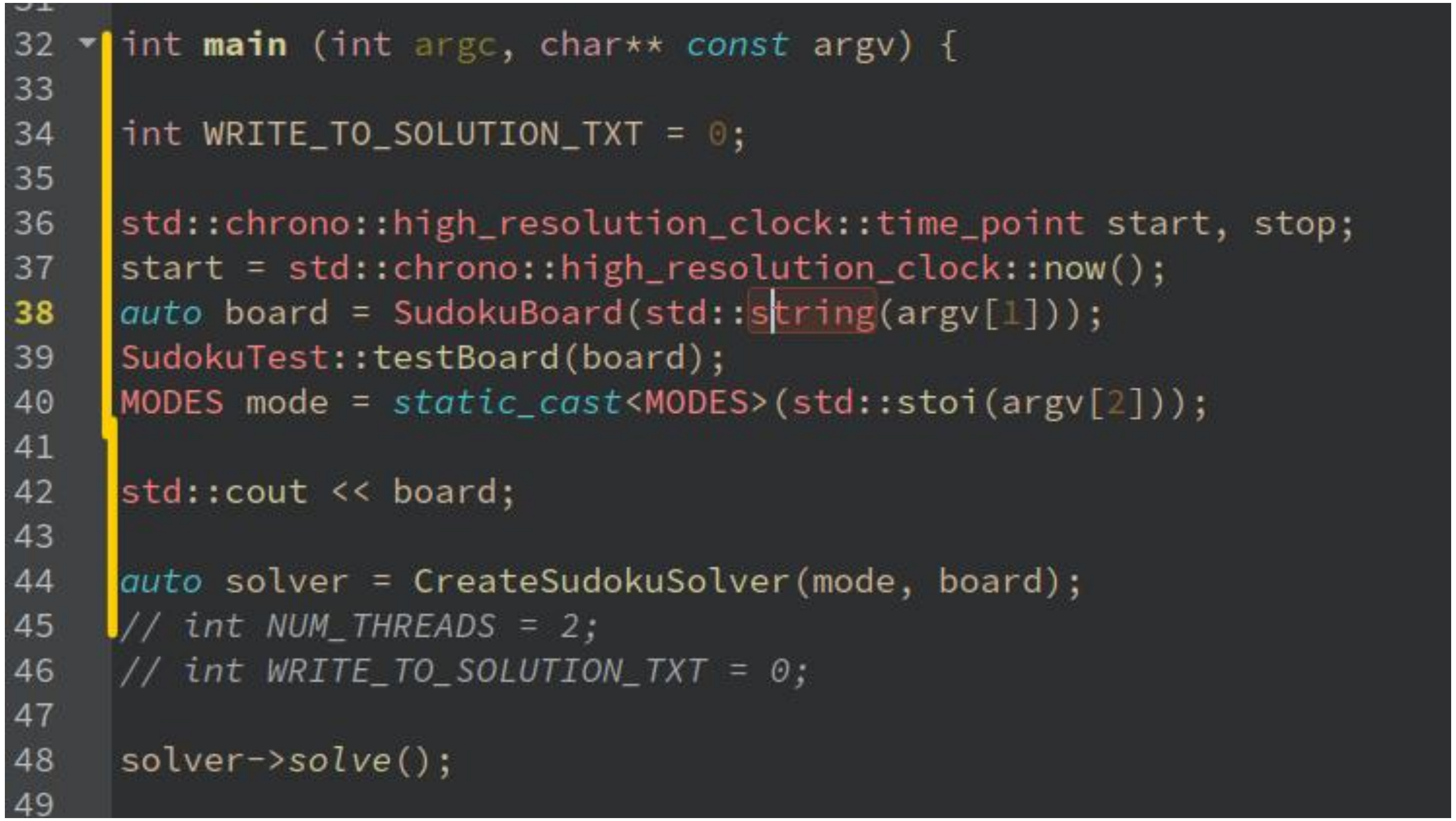


```
public:
    //@mrz: meaningless
    const Board read_input(const std::string& filename);
    // Writes solution to a text file (solution.txt)
```

https://www.linuxtopia.org/online_books/programming_books/thinking_in_c++/Chapter08_014.html

Indentation

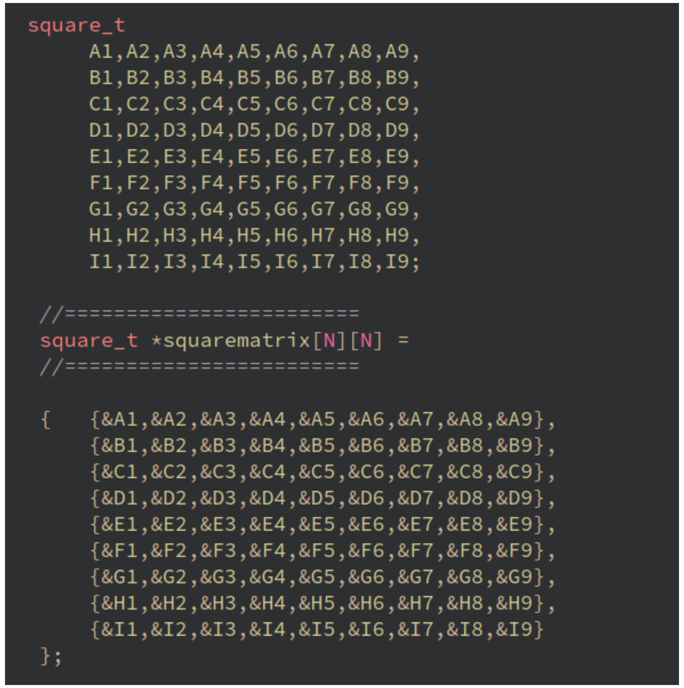
- Your code is not yours, and it is your job to make it readable for others!



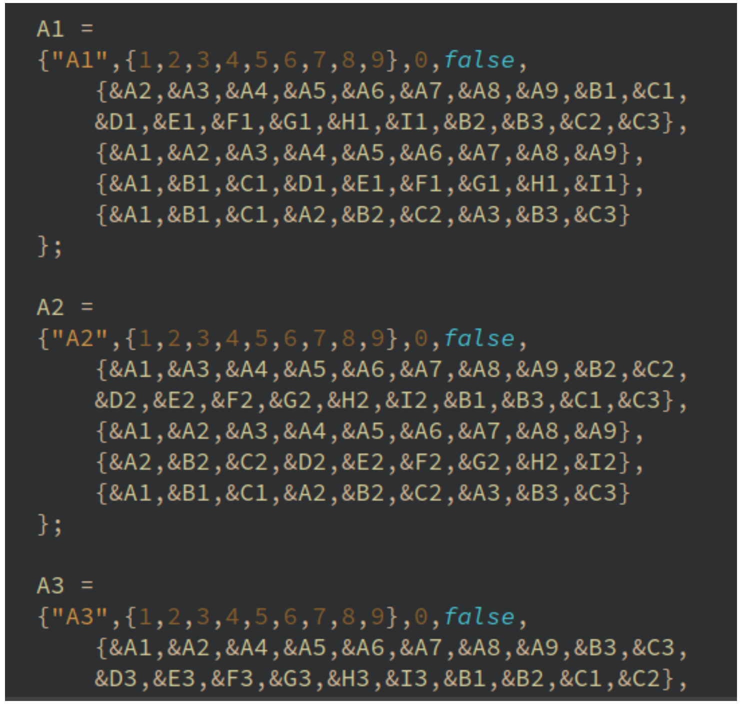
```cpp
int main (int argc, char** const argv) {

int WRITE_TO_SOLUTION_TXT = 0;

std::chrono::high_resolution_clock::time_point start, stop;
start = std::chrono::high_resolution_clock::now();
auto board = SudokuBoard(std::string(argv[1]));
SudokuTest::testBoard(board);
MODES mode = static_cast<MODES>(std::stoi(argv[2]));

std::cout << board;

auto solver = CreateSudokuSolver(mode, board);
// int NUM_THREADS = 2;
// int WRITE_TO_SOLUTION_TXT = 0;

solver->solve();

```

init

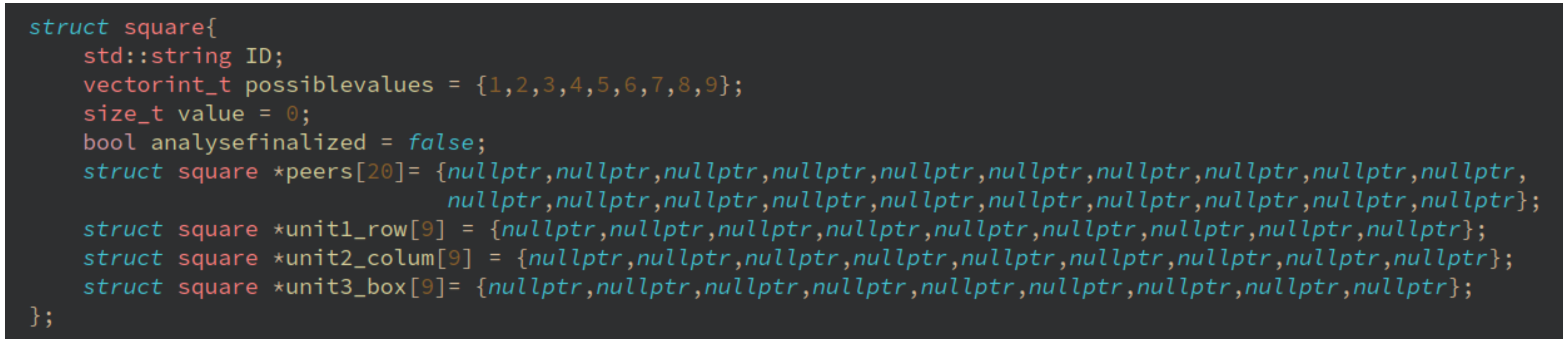
- Nice idea,
- Not very nice implementation
- 81 initialization
- Automate it – names are for human!



```
square_t
     A1,A2,A3,A4,A5,A6,A7,A8,A9,
     B1,B2,B3,B4,B5,B6,B7,B8,B9,
     C1,C2,C3,C4,C5,C6,C7,C8,C9,
     D1,D2,D3,D4,D5,D6,D7,D8,D9,
     E1,E2,E3,E4,E5,E6,E7,E8,E9,
     F1,F2,F3,F4,F5,F6,F7,F8,F9,
     G1,G2,G3,G4,G5,G6,G7,G8,G9,
     H1,H2,H3,H4,H5,H6,H7,H8,H9,
     I1,I2,I3,I4,I5,I6,I7,I8,I9;

 //========================
 square_t *squarematrix[N][N] =
 //========================

 {   {&A1,&A2,&A3,&A4,&A5,&A6,&A7,&A8,&A9},
     {&B1,&B2,&B3,&B4,&B5,&B6,&B7,&B8,&B9},
     {&C1,&C2,&C3,&C4,&C5,&C6,&C7,&C8,&C9},
     {&D1,&D2,&D3,&D4,&D5,&D6,&D7,&D8,&D9},
     {&E1,&E2,&E3,&E4,&E5,&E6,&E7,&E8,&E9},
     {&F1,&F2,&F3,&F4,&F5,&F6,&F7,&F8,&F9},
     {&G1,&G2,&G3,&G4,&G5,&G6,&G7,&G8,&G9},
     {&H1,&H2,&H3,&H4,&H5,&H6,&H7,&H8,&H9},
     {&I1,&I2,&I3,&I4,&I5,&I6,&I7,&I8,&I9}
 };
```



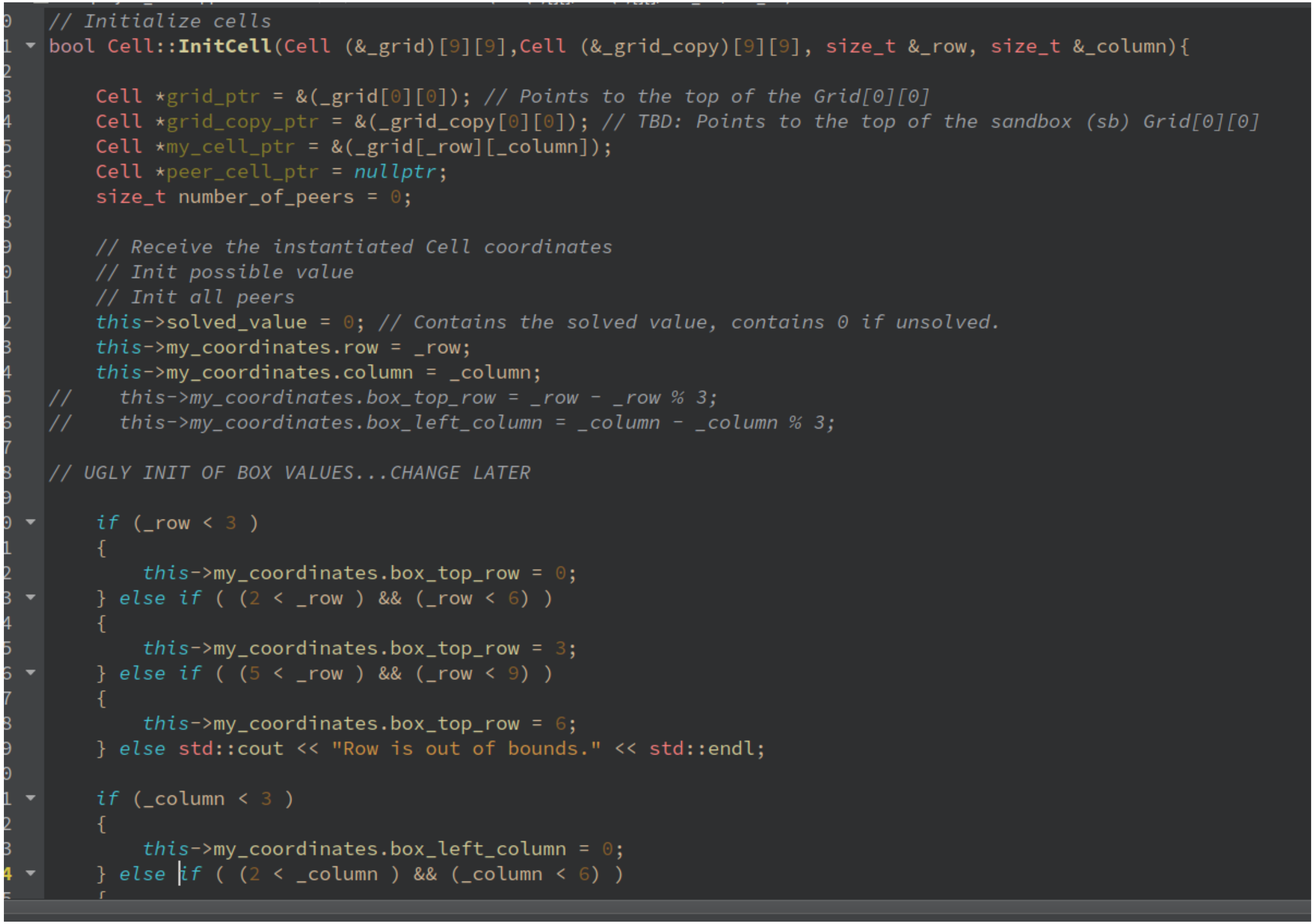
```
A1 =
{"A1",{1,2,3,4,5,6,7,8,9},0,false,
    {&A2,&A3,&A4,&A5,&A6,&A7,&A8,&A9,&B1,&C1,
    &D1,&E1,&F1,&G1,&H1,&I1,&B2,&B3,&C2,&C3},
    {&A1,&A2,&A3,&A4,&A5,&A6,&A7,&A8,&A9},
    {&A1,&B1,&C1,&D1,&E1,&F1,&G1,&H1,&I1},
    {&A1,&B1,&C1,&A2,&B2,&C2,&A3,&B3,&C3}
};

A2 =
{"A2",{1,2,3,4,5,6,7,8,9},0,false,
    {&A1,&A3,&A4,&A5,&A6,&A7,&A8,&A9,&B2,&C2,
    &D2,&E2,&F2,&G2,&H2,&I2,&B1,&B3,&C1,&C3},
    {&A1,&A2,&A3,&A4,&A5,&A6,&A7,&A8,&A9},
    {&A2,&B2,&C2,&D2,&E2,&F2,&G2,&H2,&I2},
    {&A1,&B1,&C1,&A2,&B2,&C2,&A3,&B3,&C3}
};

A3 =
{"A3",{1,2,3,4,5,6,7,8,9},0,false,
    {&A1,&A2,&A4,&A5,&A6,&A7,&A8,&A9,&B3,&C3,
    &D3,&E3,&F3,&G3,&H3,&I3,&B1,&B2,&C1,&C2},
```



```
struct square{
    std::string ID;
    vectorint_t possiblevalues = {1,2,3,4,5,6,7,8,9};
    size_t value = 0;
    bool analysefinalized = false;
    struct square *peers[20]= {nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,
                               nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr};
    struct square *unit1_row[9] = {nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr};
    struct square *unit2_colum[9] = {nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr};
    struct square *unit3_box[9]= {nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr};
};
```

init

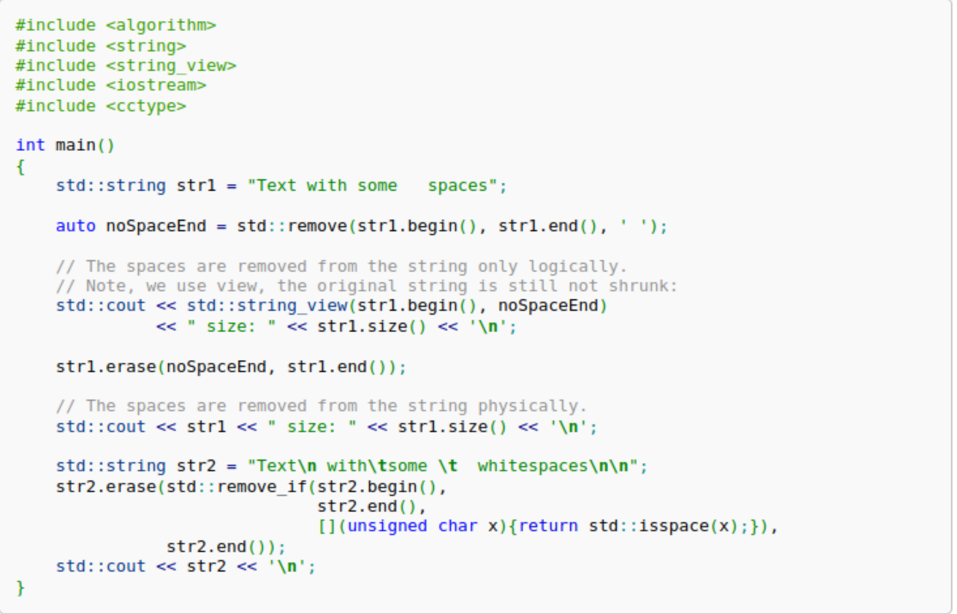
- Nice initialization!



```
// Initialize cells
bool Cell::InitCell(Cell (&_grid)[9][9],Cell (&_grid_copy)[9][9], size_t &_row, size_t &_column){

    Cell *grid_ptr = &(_grid[0][0]); // Points to the top of the Grid[0][0]
    Cell *grid_copy_ptr = &(_grid_copy[0][0]); // TBD: Points to the top of the sandbox (sb) Grid[0][0]
    Cell *my_cell_ptr = &(_grid[_row][_column]);
    Cell *peer_cell_ptr = nullptr;
    size_t number_of_peers = 0;

    // Receive the instantiated Cell coordinates
    // Init possible value
    // Init all peers
    this->solved_value = 0; // Contains the solved value, contains 0 if unsolved.
    this->my_coordinates.row = _row;
    this->my_coordinates.column = _column;
//    this->my_coordinates.box_top_row = _row - _row % 3;
//    this->my_coordinates.box_left_column = _column - _column % 3;

// UGLY INIT OF BOX VALUES...CHANGE LATER

    if (_row < 3 )
    {
        this->my_coordinates.box_top_row = 0;
    } else if ( (2 < _row ) && (_row < 6) )
    {
        this->my_coordinates.box_top_row = 3;
    } else if ( (5 < _row ) && (_row < 9) )
    {
        this->my_coordinates.box_top_row = 6;
    } else std::cout << "Row is out of bounds." << std::endl;

    if (_column < 3 )
    {
        this->my_coordinates.box_left_column = 0;
    } else if ( (2 < _column ) && (_column < 6) )
    {
```

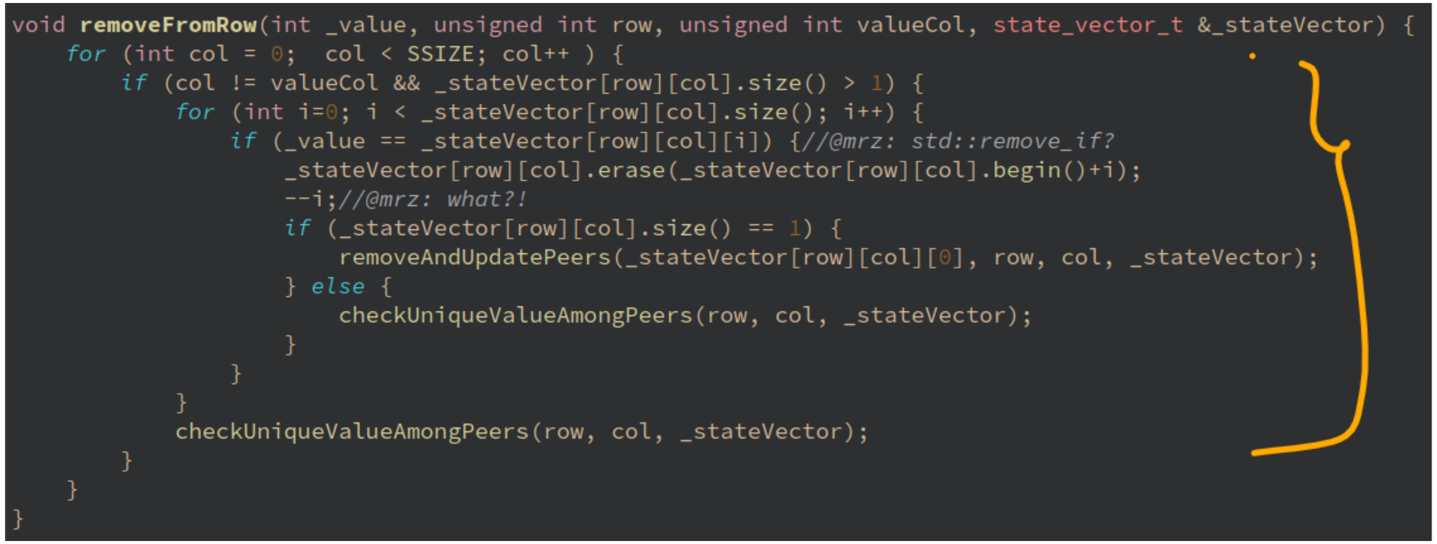
Vector earase

© M. Rashid Zamani

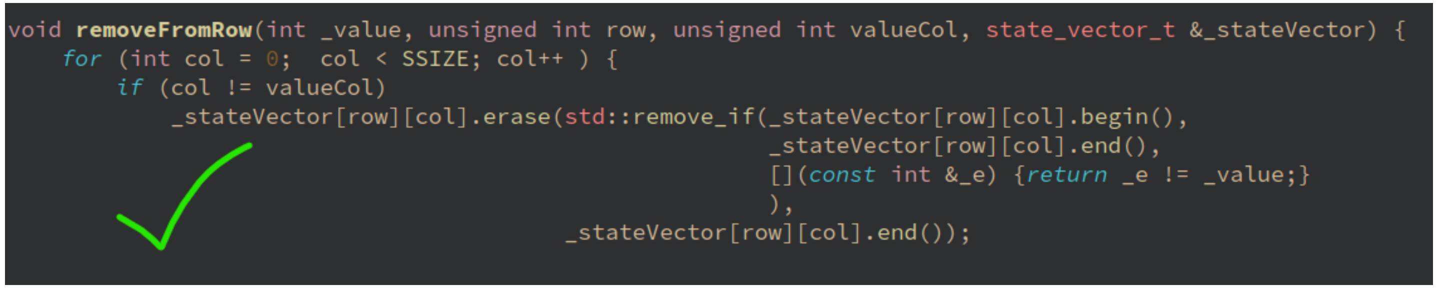


```cpp
#include <algorithm>
#include <string>
#include <string_view>
#include <iostream>
#include <cctype>

int main()
{
    std::string str1 = "Text with some   spaces";

    auto noSpaceEnd = std::remove(str1.begin(), str1.end(), ' ');

    // The spaces are removed from the string only logically.
    // Note, we use view, the original string is still not shrunk:
    std::cout << std::string_view(str1.begin(), noSpaceEnd)
              << " size: " << str1.size() << '\n';

    str1.erase(noSpaceEnd, str1.end());

    // The spaces are removed from the string physically.
    std::cout << str1 << " size: " << str1.size() << '\n';

    std::string str2 = "Text\n with\tsome \t  whitespaces\n\n";
    str2.erase(std::remove_if(str2.begin(),
                              str2.end(),
                              [](unsigned char x){return std::isspace(x);}),
               str2.end());
    std::cout << str2 << '\n';
}
```

https://en.cppreference.com/w/cpp/algorithm/remove



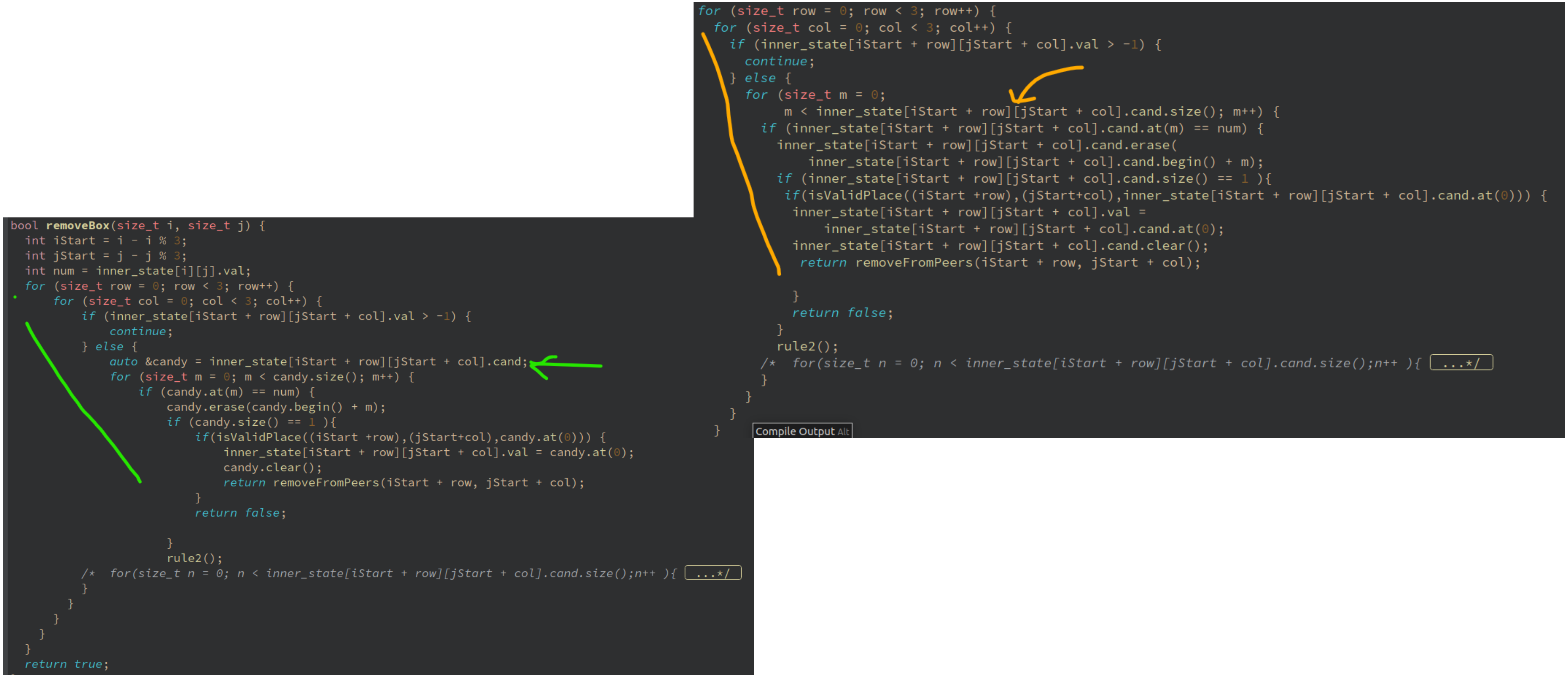
```cpp
void removeFromRow(int _value, unsigned int row, unsigned int valueCol, state_vector_t &_stateVector) {
    for (int col = 0;  col < SSIZE; col++ ) {
        if (col != valueCol && _stateVector[row][col].size() > 1) {
            for (int i=0; i < _stateVector[row][col].size(); i++) {
                if (_value == _stateVector[row][col][i]) {//@mrz: std::remove_if?
                    _stateVector[row][col].erase(_stateVector[row][col].begin()+i);
                    --i;//@mrz: what?!
                    if (_stateVector[row][col].size() == 1) {
                        removeAndUpdatePeers(_stateVector[row][col][0], row, col, _stateVector);
                    } else {
                        checkUniqueValueAmongPeers(row, col, _stateVector);
                    }
                }
            }
            checkUniqueValueAmongPeers(row, col, _stateVector);
        }
    }
}
```



```cpp
void removeFromRow(int _value, unsigned int row, unsigned int valueCol, state_vector_t &_stateVector) {
    for (int col = 0;  col < SSIZE; col++ ) {
        if (col != valueCol)
            _stateVector[row][col].erase(std::remove_if(_stateVector[row][col].begin(),
                                                        _stateVector[row][col].end(),
                                                        [](const int &_e) {return _e != _value;}
                                                        ),
                                         _stateVector[row][col].end());
```

Sensitivity: C2-Restricted

Use auto!

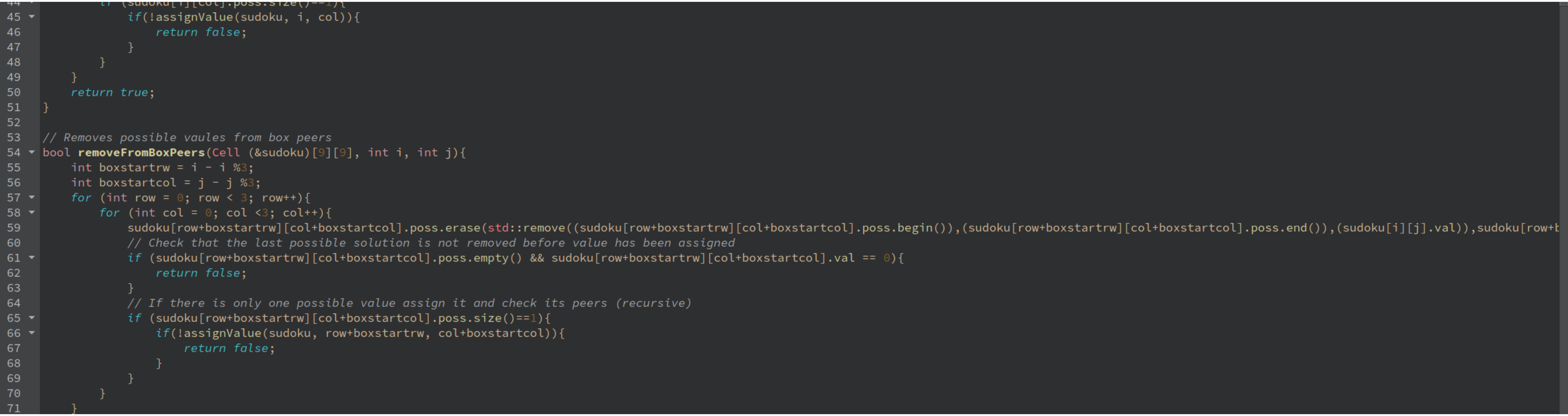


```
for (size_t row = 0; row < 3; row++) {
  for (size_t col = 0; col < 3; col++) {
    if (inner_state[iStart + row][jStart + col].val > -1) {
      continue;
    } else {
      for (size_t m = 0;
           m < inner_state[iStart + row][jStart + col].cand.size(); m++) {
        if (inner_state[iStart + row][jStart + col].cand.at(m) == num) {
          inner_state[iStart + row][jStart + col].cand.erase(
              inner_state[iStart + row][jStart + col].cand.begin() + m);
          if (inner_state[iStart + row][jStart + col].cand.size() == 1 ){
           if(isValidPlace((iStart +row),(jStart+col),inner_state[iStart + row][jStart + col].cand.at(0))) {
            inner_state[iStart + row][jStart + col].val =
                inner_state[iStart + row][jStart + col].cand.at(0);
            inner_state[iStart + row][jStart + col].cand.clear();
             return removeFromPeers(iStart + row, jStart + col);

            }
            return false;
          }
          rule2();
        /*  for(size_t n = 0; n < inner_state[iStart + row][jStart + col].cand.size();n++ ){ ...*/
        }
      }
    }
  }
```

```
bool removeBox(size_t i, size_t j) {
  int iStart = i - i % 3;
  int jStart = j - j % 3;
  int num = inner_state[i][j].val;
  for (size_t row = 0; row < 3; row++) {
      for (size_t col = 0; col < 3; col++) {
          if (inner_state[iStart + row][jStart + col].val > -1) {
              continue;
          } else {
              auto &candy = inner_state[iStart + row][jStart + col].cand;
              for (size_t m = 0; m < candy.size(); m++) {
                  if (candy.at(m) == num) {
                      candy.erase(candy.begin() + m);
                      if (candy.size() == 1 ){
                          if(isValidPlace((iStart +row),(jStart+col),candy.at(0))) {
                              inner_state[iStart + row][jStart + col].val = candy.at(0);
                              candy.clear();
                              return removeFromPeers(iStart + row, jStart + col);
                          }
                          return false;

                      }
                      rule2();
          /*  for(size_t n = 0; n < inner_state[iStart + row][jStart + col].cand.size();n++ ){ ...*/
          }
        }
      }
    }
  }
  return true;
```

Break the line

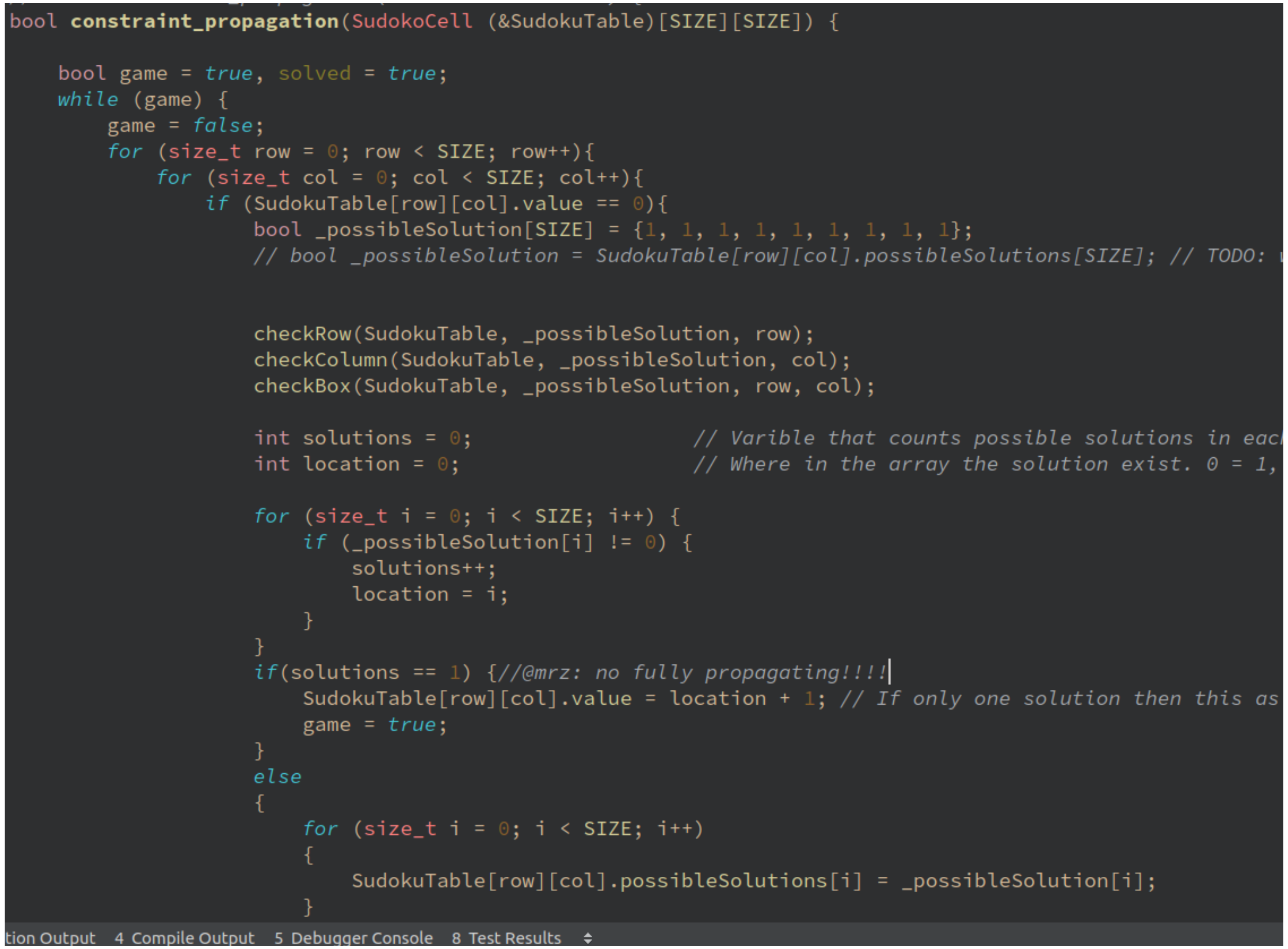
- Break the line! Certain width is allowed (120 characters usually)



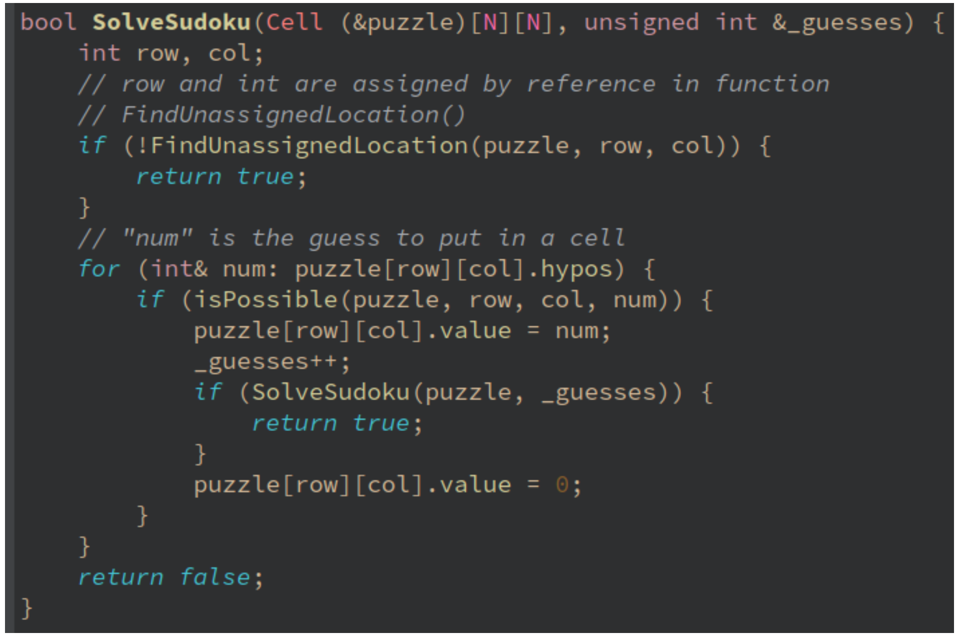
```
        if (sudoku[i][col].poss.size()==1){
            if(!assignValue(sudoku, i, col)){
                return false;
            }
        }
    }
    return true;
}

// Removes possible vaules from box peers
bool removeFromBoxPeers(Cell (&sudoku)[9][9], int i, int j){
    int boxstartrw = i - i %3;
    int boxstartcol = j - j %3;
    for (int row = 0; row < 3; row++){
        for (int col = 0; col <3; col++){
            sudoku[row+boxstartrw][col+boxstartcol].poss.erase(std::remove((sudoku[row+boxstartrw][col+boxstartcol].poss.begin()),(sudoku[row+boxstartrw][col+boxstartcol].poss.end()),(sudoku[i][j].val)),sudoku[row+b
            // Check that the last possible solution is not removed before value has been assigned
            if (sudoku[row+boxstartrw][col+boxstartcol].poss.empty() && sudoku[row+boxstartrw][col+boxstartcol].val == 0){
                return false;
            }
            // If there is only one possible value assign it and check its peers (recursive)
            if (sudoku[row+boxstartrw][col+boxstartcol].poss.size()==1){
                if(!assignValue(sudoku, row+boxstartrw, col+boxstartcol)){
                    return false;
                }
            }
        }
    }
```

Search

- Stateless search!
- No propagation during search!



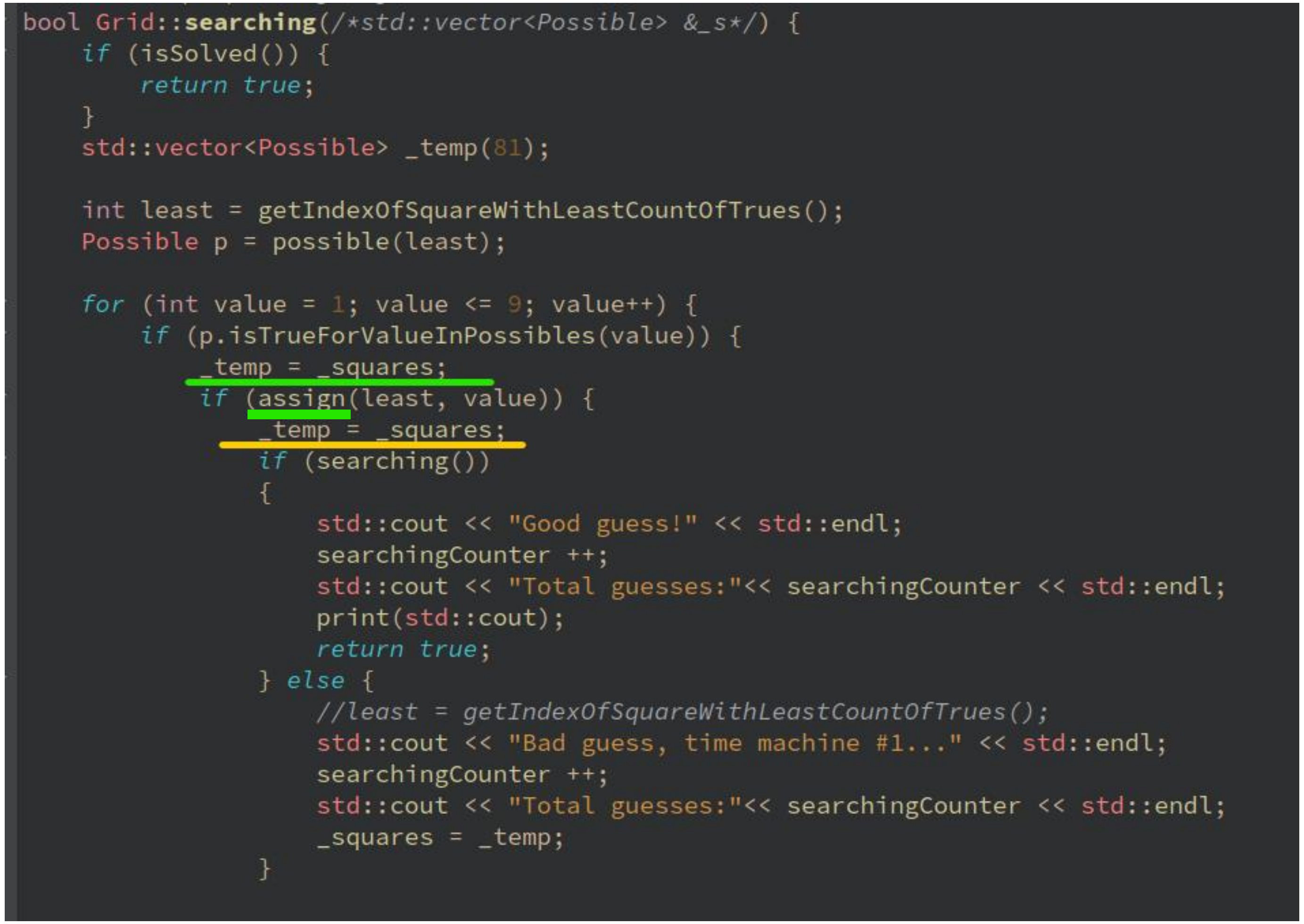
```
bool constraint_propagation(SudokoCell (&SudokuTable)[SIZE][SIZE]) {

    bool game = true, solved = true;
    while (game) {
        game = false;
        for (size_t row = 0; row < SIZE; row++){
            for (size_t col = 0; col < SIZE; col++){
                if (SudokuTable[row][col].value == 0){
                    bool _possibleSolution[SIZE] = {1, 1, 1, 1, 1, 1, 1, 1, 1};
                    // bool _possibleSolution = SudokuTable[row][col].possibleSolutions[SIZE]; // TODO:


                    checkRow(SudokuTable, _possibleSolution, row);
                    checkColumn(SudokuTable, _possibleSolution, col);
                    checkBox(SudokuTable, _possibleSolution, row, col);

                    int solutions = 0;                    // Varible that counts possible solutions in eac
                    int location = 0;                     // Where in the array the solution exist. 0 = 1,

                    for (size_t i = 0; i < SIZE; i++) {
                        if (_possibleSolution[i] != 0) {
                            solutions++;
                            location = i;
                        }
                    }
                    if(solutions == 1) {//@mrz: no fully propagating!!!!
                        SudokuTable[row][col].value = location + 1; // If only one solution then this as
                        game = true;
                    }
                    else
                    {
                        for (size_t i = 0; i < SIZE; i++)
                        {
                            SudokuTable[row][col].possibleSolutions[i] = _possibleSolution[i];
                        }
```



```
bool SolveSudoku(Cell (&puzzle)[N][N], unsigned int &_guesses) {
    int row, col;
    // row and int are assigned by reference in function
    // FindUnassignedLocation()
    if (!FindUnassignedLocation(puzzle, row, col)) {
        return true;
    }
    // "num" is the guess to put in a cell
    for (int& num: puzzle[row][col].hypos) {
        if (isPossible(puzzle, row, col, num)) {
            puzzle[row][col].value = num;
            _guesses++;
            if (SolveSudoku(puzzle, _guesses)) {
                return true;
            }
            puzzle[row][col].value = 0;
        }
    }
    return false;
}
```

Search

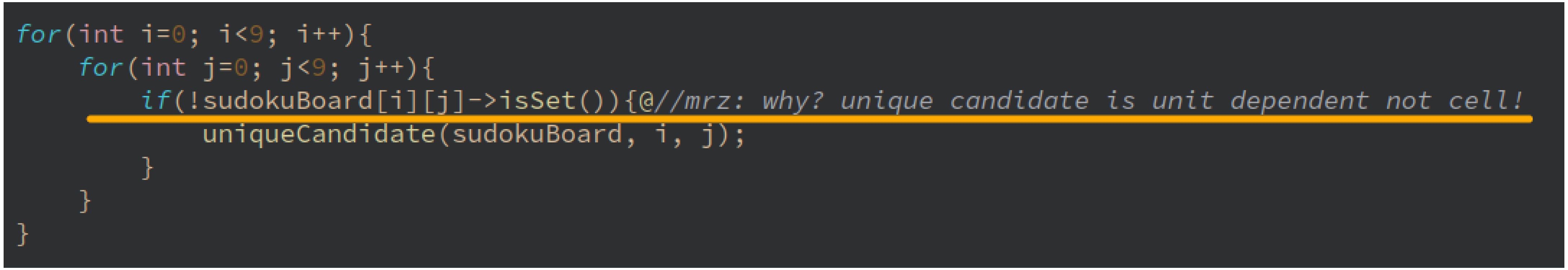
- Nice copy
- Assign to propagate fully



```
bool Grid::searching(/*std::vector<Possible> &_s*/) {
  if (isSolved()) {
    return true;
  }
  std::vector<Possible> _temp(81);

  int least = getIndexOfSquareWithLeastCountOfTrues();
  Possible p = possible(least);

  for (int value = 1; value <= 9; value++) {
    if (p.isTrueForValueInPossibles(value)) {
      _temp = _squares;
      if (assign(least, value)) {
        _temp = _squares;
        if (searching())
        {
          std::cout << "Good guess!" << std::endl;
          searchingCounter ++;
          std::cout << "Total guesses:"<< searchingCounter << std::endl;
          print(std::cout);
          return true;
        } else {
          //least = getIndexOfSquareWithLeastCountOfTrues();
          std::cout << "Bad guess, time machine #1..." << std::endl;
          searchingCounter ++;
          std::cout << "Total guesses:"<< searchingCounter << std::endl;
          _squares = _temp;
        }
```

Rule 2

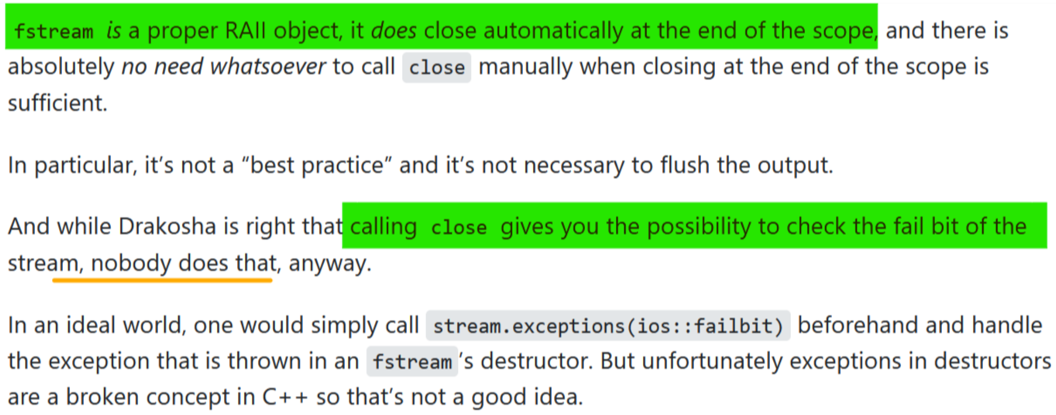
- Rule 2 is unit dependent not square (cell)!



```
for(int i=0; i<9; i++){
    for(int j=0; j<9; j++){
        if(!sudokuBoard[i][j]->isSet()){@//mrz: why? unique candidate is unit dependent not cell!
            uniqueCandidate(sudokuBoard, i, j);
        }
    }
}
```

RAII

- Close files!



`fstream` *is* a proper RAII object, it *does* close automatically at the end of the scope, and there is absolutely *no need whatsoever* to call `close` manually when closing at the end of the scope is sufficient.

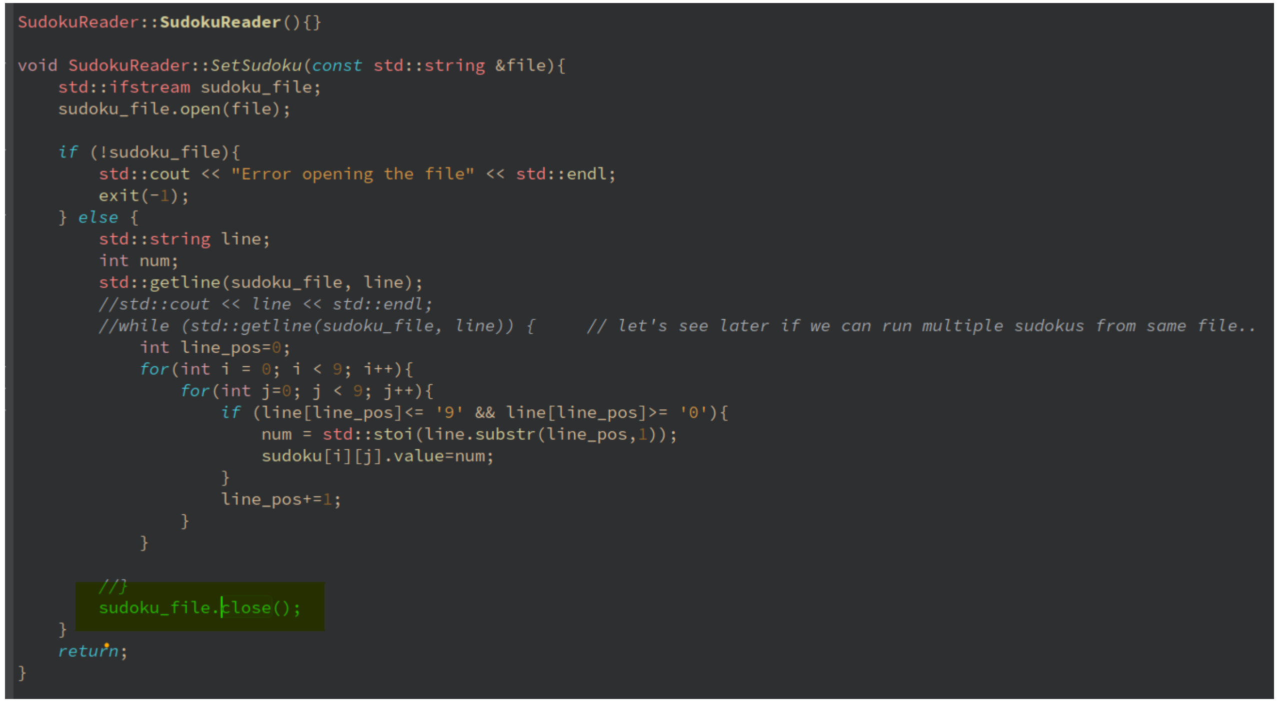
In particular, it's not a "best practice" and it's not necessary to flush the output.

And while Drakosha is right that calling `close` gives you the possibility to check the fail bit of the stream, nobody does that, anyway.

In an ideal world, one would simply call `stream.exceptions(ios::failbit)` beforehand and handle the exception that is thrown in an `fstream`'s destructor. But unfortunately exceptions in destructors are a broken concept in C++ so that's not a good idea.

So **if** you want to check the success of closing a file, do it manually (but only then).

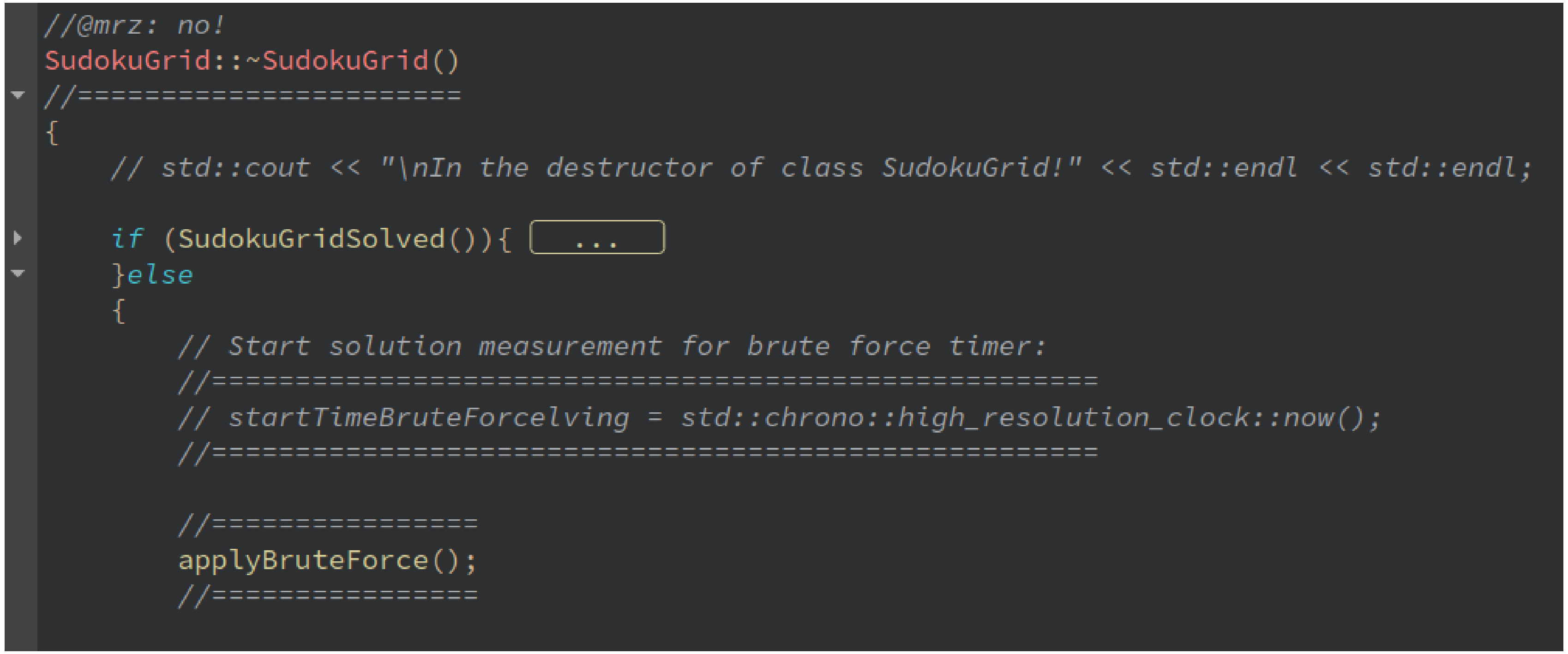
https://stackoverflow.com/questions/4802494/do-i-need-to-close-a-stdfstream



```
SudokuReader::SudokuReader(){}

void SudokuReader::SetSudoku(const std::string &file){
    std::ifstream sudoku_file;
    sudoku_file.open(file);

    if (!sudoku_file){
        std::cout << "Error opening the file" << std::endl;
        exit(-1);
    } else {
        std::string line;
        int num;
        std::getline(sudoku_file, line);
        //std::cout << line << std::endl;
        //while (std::getline(sudoku_file, line)) {     // let's see later if we can run multiple sudokus from same file..
            int line_pos=0;
            for(int i = 0; i < 9; i++){
                for(int j=0; j < 9; j++){
                    if (line[line_pos]<= '9' && line[line_pos]>= '0'){
                        num = std::stoi(line.substr(line_pos,1));
                        sudoku[i][j].value=num;
                    }
                    line_pos+=1;
                }
            }

        //}
        sudoku_file.close();
    }
    return;
}
```

~

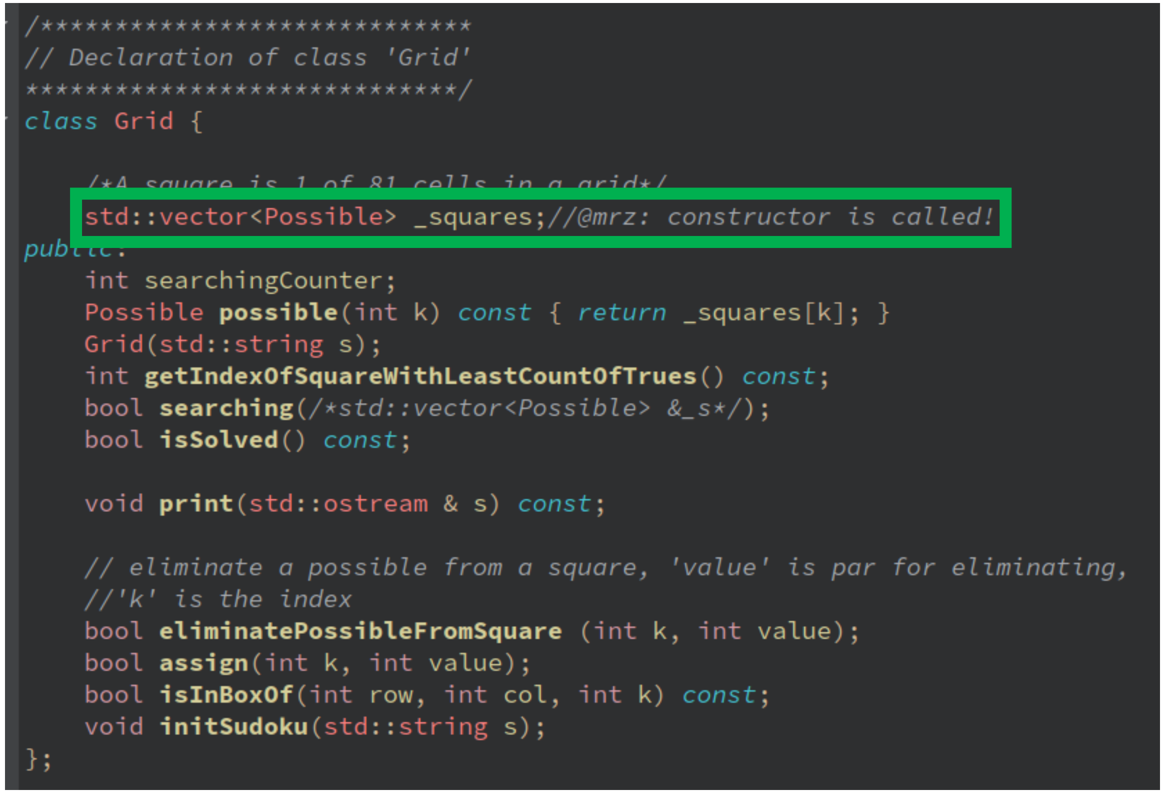
- Destructor is only to clean up!!!!



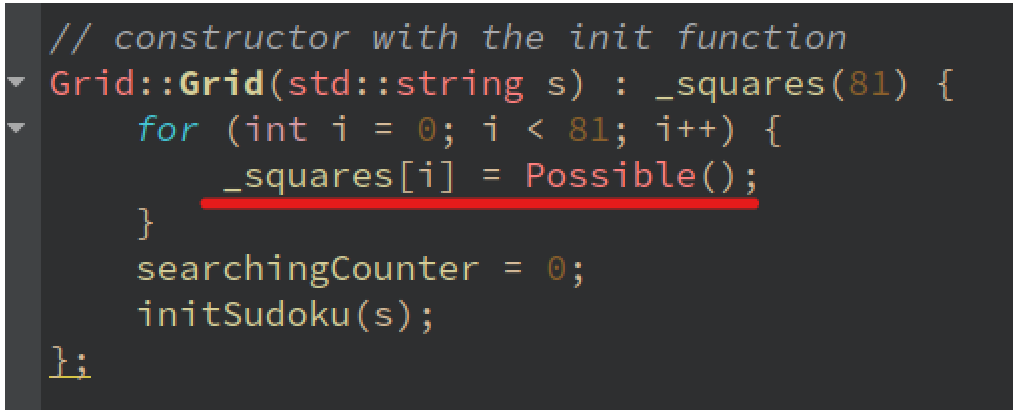
```
//@mrz: no!
SudokuGrid::~SudokuGrid()
//=========================
{
    // std::cout << "\nIn the destructor of class SudokuGrid!" << std::endl << std::endl;

    if (SudokuGridSolved()){ ...
    }else
    {
        // Start solution measurement for brute force timer:
        //=====================================================
        // startTimeBruteForcelving = std::chrono::high_resolution_clock::now();
        //=====================================================

        //=================
        applyBruteForce();
        //=================
```

GOD

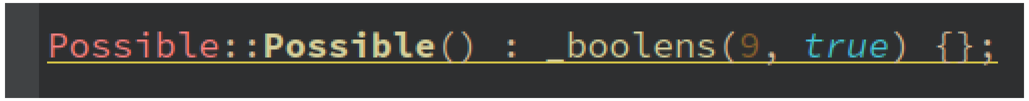
- The squares(state) is tied to grid, and the logic – grid is dead, what if another logic needs to be applied to your grid?! Grid GOD object!



```
/****************************
// Declaration of class 'Grid'
****************************/
class Grid {

    /*A square is 1 of 81 cells in a grid*/
    std::vector<Possible> _squares;//@mrz: constructor is called!
public:
    int searchingCounter;
    Possible possible(int k) const { return _squares[k]; }
    Grid(std::string s);
    int getIndexOfSquareWithLeastCountOfTrues() const;
    bool searching(/*std::vector<Possible> &_s*/);
    bool isSolved() const;

    void print(std::ostream & s) const;

    // eliminate a possible from a square, 'value' is par for eliminating,
    //'k' is the index
    bool eliminatePossibleFromSquare (int k, int value);
    bool assign(int k, int value);
    bool isInBoxOf(int row, int col, int k) const;
    void initSudoku(std::string s);
};
```



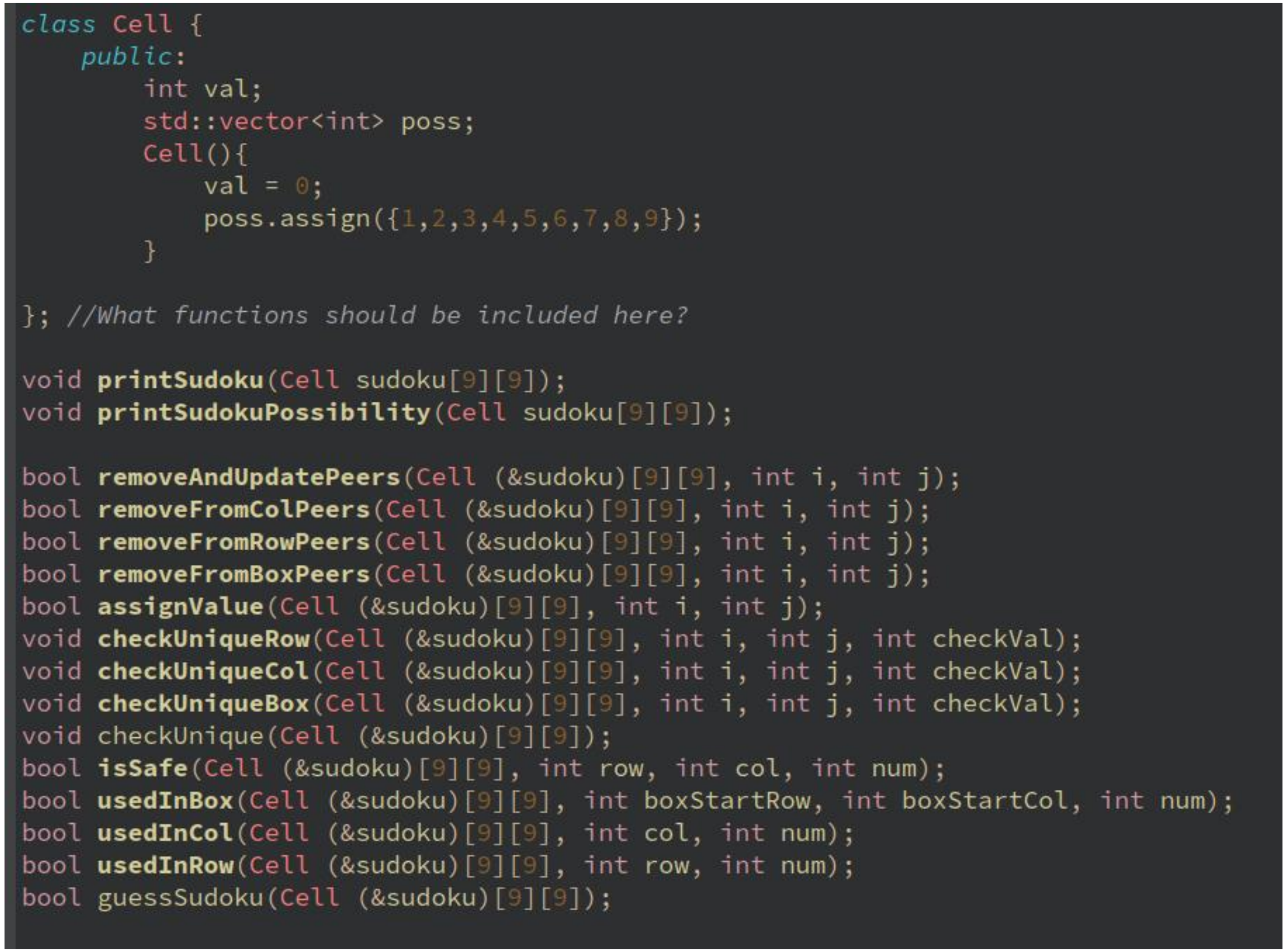
```
// constructor with the init function
Grid::Grid(std::string s) : _squares(81) {
    for (int i = 0; i < 81; i++) {
        _squares[i] = Possible();
    }
    searchingCounter = 0;
    initSudoku(s);
};
```



```
Possible::Possible() : _boolens(9, true) {};
```

Less is more!

- Nice separation!



```cpp
class Cell {
    public:
        int val;
        std::vector<int> poss;
        Cell(){
            val = 0;
            poss.assign({1,2,3,4,5,6,7,8,9});
        }

}; //What functions should be included here?

void printSudoku(Cell sudoku[9][9]);
void printSudokuPossibility(Cell sudoku[9][9]);

bool removeAndUpdatePeers(Cell (&sudoku)[9][9], int i, int j);
bool removeFromColPeers(Cell (&sudoku)[9][9], int i, int j);
bool removeFromRowPeers(Cell (&sudoku)[9][9], int i, int j);
bool removeFromBoxPeers(Cell (&sudoku)[9][9], int i, int j);
bool assignValue(Cell (&sudoku)[9][9], int i, int j);
void checkUniqueRow(Cell (&sudoku)[9][9], int i, int j, int checkVal);
void checkUniqueCol(Cell (&sudoku)[9][9], int i, int j, int checkVal);
void checkUniqueBox(Cell (&sudoku)[9][9], int i, int j, int checkVal);
void checkUnique(Cell (&sudoku)[9][9]);
bool isSafe(Cell (&sudoku)[9][9], int row, int col, int num);
bool usedInBox(Cell (&sudoku)[9][9], int boxStartRow, int boxStartCol, int num);
bool usedInCol(Cell (&sudoku)[9][9], int col, int num);
bool usedInRow(Cell (&sudoku)[9][9], int row, int num);
bool guessSudoku(Cell (&sudoku)[9][9]);
```

This is FINE!

© M. Rashid Zamani

THIS IS FINE.
I'M OKAY WITH THE EVENTS THAT ARE UNFOLDING CURRENTLY.
THAT'S OKAY, THINGS ARE GOING TO BE OKAY.