cs280s21-b.sg

Dashboard / My courses / cs280s21-b.sg / General / Bonus Assignment: Sudoku

Description | Submission view

Grade

Reviewed on Sunday, March 14, 2021, 2:17 AM by Automatic grade
grade: 100.00 / 100.00

Assessment report[-]
[+]**Failed tests**
[+]**Test 10: board4-3**
[+]**Test 13: board5-2**
[+]**Summary of tests**

Submitted on Sunday, March 14, 2021, 2:11 AM (Download)

Sudoku.h

```
/******************************************************************************/
/*!
\file:      Sudoku.h
\author:    Goh Wei Zhe, weizhe.goh, 440000119
\par email: weizhe.goh\@digipen.edu
\date:      March 8, 2021
\brief      This file contains the declarations needed to implement a simple
            recursive algorithm to solve a sudoku puzzle.

Copyright (C) 2021 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/******************************************************************************/

//---------------------------------------------------------------------------
#ifndef SUDOKUH
#define SUDOKUH
//---------------------------------------------------------------------------
#include <cstddef> /* size_t */

//! The Sudoku class
class Sudoku
{
  public:
    //! Used by the callback function
    enum MessageType
    {
      MSG_STARTING,      //!< the board is setup, ready to go
      MSG_FINISHED_OK,   //!< finished and found a solution
      MSG_FINISHED_FAIL, //!< finished but no solution found
      MSG_ABORT_CHECK,   //!< checking to see if algorithm should continue
      MSG_PLACING,       //!< placing a symbol on the board
      MSG_REMOVING       //!< removing a symbol (back-tracking)
    };

    //! 1-9 for 9x9, A-P for 16x16, A-Y for 25x25
    enum SymbolType {SYM_NUMBER, SYM_LETTER};

    //! Represents an empty cell (the driver will use a . instead)
    const static char EMPTY_CHAR = ' ';

    //! Implemented in the client and called during the search for a solution
    typedef bool (*SUDOKU_CALLBACK)
      (const Sudoku& sudoku, // the gameboard object itself
       const char *board,    // one-dimensional array of symbols
       MessageType message,  // type of message
       size_t move,          // the move number
       unsigned basesize,    // 3, 4, 5, etc. (for 9x9, 16x16, 25x25, etc.)
       unsigned index,       // index (0-based) of current cell
       char value            // symbol (value) in current cell
      );

    //! Statistics as the algorithm works
    struct SudokuStats
    {
      int basesize;      //!< 3, 4, 5, etc.
      int placed;        //!< number of valid values the algorithm has placed
      size_t moves;      //!< total number of values that have been tried
      size_t backtracks; //!< total number of times the algorithm backtracked

      //!< Default constructor
      SudokuStats() : basesize(0), placed(0), moves(0), backtracks(0) {}
    };

      // Constructor
    Sudoku(int basesize, SymbolType stype = SYM_NUMBER,
            SUDOKU_CALLBACK callback = 0);

      // Destructor
    ~Sudoku();

      // The client (driver) passed the board in the values parameter
    void SetupBoard(const char *values, int size);

      // Once the board is setup, this will start the search for the solution
    void Solve();

      // For debugging with the driver
    const char *GetBoard() const;
    SudokuStats GetStats() const;

  private:
  // Other private data members or methods...

    int board_width;
    int board_size;

    char* board_;
    SudokuStats stats_;

    SymbolType stype_;
    SUDOKU_CALLBACK callback_;

    bool place_value(int value);
    bool Conflict(int index, char value);
};

#endif  // SUDOKUH

```

Sudoku.cpp

```cpp
/******************************************************************************/
/*!
\file:      Sudoku.cpp
\author:    Goh Wei Zhe, weizhe.goh, 440000119
\par email: weizhe.goh\@digipen.edu
\date:      March 8, 2021
\brief      This file contains the definition needed to implement a simple
            recursive algorithm to solve a sudoku puzzle.

Copyright (C) 2021 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/******************************************************************************/
#include "Sudoku.h"

/******************************************************************************/
/*!
\fn     Sudoku::Sudoku(int basesize, SymbolType stype, SUDOKU_CALLBACK callback)
        :stype_{stype}, callback_{callback}

\param  basesize - The base size of sudoku board

\param  stype - The type of symbol used for the sudoku puzzle, number or letter

\param  callback - The particular type of callback

\brief  Constructor for sudoku
*/
/******************************************************************************/
Sudoku::Sudoku(int basesize, SymbolType stype, SUDOKU_CALLBACK callback)
:stype_{stype}, callback_{callback}
{
    stats_.basesize = basesize;

    board_width = basesize * basesize;
    board_size = board_width * board_width;
}

/******************************************************************************/
/*!
\fn     Sudoku::~Sudoku()

\brief  Destructor for sudoku, delete board
*/
/******************************************************************************/
Sudoku::~Sudoku()
{
    delete[] board_;
}

/******************************************************************************/
/*!
\fn     const char* Sudoku::GetBoard() const

\brief  Gettor for sudoku board

\return Returns char array of values within the board
*/
/******************************************************************************/
const char* Sudoku::GetBoard() const
{
    return board_;
}

/******************************************************************************/
/*!
\fn     Sudoku::SudokuStats Sudoku::GetStats() const

\brief  Gettor for sudoku statistics

\return Returns data structure SudokuStats for sudoku board
*/
/******************************************************************************/
Sudoku::SudokuStats Sudoku::GetStats() const
{
    return stats_;
}

/******************************************************************************/
/*!
\fn     void Sudoku::SetupBoard(const char* values, int size)

\brief   Set up sodoku board with specific board values and board size

\param  values - an array of values to fill up the sudoku board

\param  size - size of sudoku board

*/
/******************************************************************************/
void Sudoku::SetupBoard(const char* values, int size)
{
    board_ = new char[size];

    //Set board values to be empty or filled with values;
    for(int i = 0; i < size; ++i)
        board_[i] = (values[i] == '.') ? EMPTY_CHAR : values[i];
}

/******************************************************************************/
/*!
\fn     void Sudoku::Solve()

\brief  Attempts to solve the sudoku board
*/
/******************************************************************************/
```

```cpp
void Sudoku::Solve()
{
    //When you start the algorithm (the client calls Solve),
    //you will send MSG_STARTING.
    callback_(*this, board_, MSG_STARTING, stats_.moves,
    stats_.basesize, 0, stype_);

    int value = 0;

    if(place_value(value))
    //If, after placing a value you have filled the board, you will send
    //MSG_FINISHED_OK and terminate the search.
        callback_(*this, board_, MSG_FINISHED_OK, stats_.moves,
                stats_.basesize, 0, stype_);
    else
    //If you do not find a solution after exhaustively checking, you will send
    //MSG_FINISHED_FAIL.
        callback_(*this, board_, MSG_FINISHED_FAIL, stats_.moves,
                stats_.basesize, 0, stype_);
}

/******************************************************************************/
/*!
\fn     bool Sudoku::place_value(int index)

\brief  Recursive function that place values cell by cell till board is
        completed or deem unsolvable.

\param  index - The position of the sudoku board to place the value at.

\return Returns true if able to place a value without conflicts. Else, return
        false.
*/
/******************************************************************************/
bool Sudoku::place_value(int index)
{
    //return if index is at end of sudoku board
    if(index == board_size)
        return true;

    //if board position is filled with other values, move to next index
    if(board_[index] != EMPTY_CHAR)
        return place_value(index + 1);

    char value;

    //set value type if its number or letter
    if(stype_ == SymbolType::SYM_NUMBER)
        value = '1';
    else
        value = 'A';

    for(int i = 0; i < board_width; ++i)
    {
        //You will send MSG_ABORT_CHECK immediately before you place a value or
        //remove a value. If this call returns true, you will terminate
        //the search.
        if(callback_(*this, board_, MSG_ABORT_CHECK, stats_.moves,
            stats_.basesize, index, value))
            return false;

        //Place value onto board
        board_[index] = value;

        //Increment moves and place count
        stats_.moves++;
        stats_.placed++;

        //After you place a value on the board, you will send MSG_PLACING.
        callback_(*this, board_, MSG_PLACING, stats_.moves,
                stats_.basesize, index, value);

        //If if there is conflict
        if(!Conflict(index, value))
        {
            //Go to next index if there is no conflict
             if(place_value(index + 1))
                return true;

            //if fail to place value, need to increment backtrack count
            stats_.backtracks++;
        }

        //if conflict, remove value by setting board index back to empty
        //decrement number of place count
        board_[index] = EMPTY_CHAR;
        stats_.placed--;

        //After removing a value from the board, you will send MSG_REMOVING.
        callback_(*this, board_, MSG_REMOVING, stats_.moves,
            stats_.basesize, index, value);

        //increment to next number or letter
        value++;
    }

    return false;
}

/******************************************************************************/
/*!
\fn     bool Sudoku::Conflict(int index, char value)

\brief  Function to check if value in cell has conflicts in row, in column or
        within box.

\param  index - The position of the cell in the sudoku board
```

```cpp
\param  value - value in cell to check if there is conflict

\return Returns true there is same value within row, column or in box. Else,
        return false.
*/
/******************************************************************************/
bool Sudoku::Conflict(int index, char value)
{
    //get row as x-axis and column as y-axis
    int x = index % board_width;
    int y = index / board_width;

    int row_start = y * board_width;

    //check row
    for(int i = row_start; i < row_start + board_width; ++i)
    {
        if(i == index)
            continue;

        if(board_[i] == value)
            return true;
    }

    //check column
    for(int i = 0; i < board_width; ++i)
    {
        int curr_pos = i * board_width + x;

        if(index == curr_pos)
            continue;

        if(board_[curr_pos] == value)
            return true;
    }

    //check box
    int startX = x - x % stats_.basesize;
    int startY = y - y % stats_.basesize;

    for(int i = 0; i < stats_.basesize; ++i)
    {
        for(int j = 0; j < stats_.basesize; ++j)
        {
            int curr_pos = ((startY + i) * board_width) + startX + j;

            if(index == curr_pos)
                continue;

            if(board_[curr_pos] == value)
                return true;
        }
    }

    return false;
```

VPL

◄ Assignment 2: B List

Jump to...

Assignment 3: AVL Trees ►