数独解题与生成器设计文档

## 编写目的

本技术文档详细描述了“数独解题生成器”提供的所有功能与算法描述，其中包括用户的功能性需求和非功能性需求。同时为设计与开发者提供参考。

本文档的预期读者是设计与开发人员、用户等。

## 背景

1. 待开发软件名称：数独解题与生成器。
2. 本项目的任务提出者：汤世平。
3. 本项目的设计人员：周赫斌。
4. 本项目的开发人员：周赫斌
5. 本项目的用户：未定。

## 术语

DLX , Dancing Links X ：基于循环十字链表的精确覆盖问题算法。

## 参考文献

“肥宅大哭.jpg”队大学生程序设计竞赛算法模板(2.0版)

## 需求设计

1. 功能需求

软件需要实现的功能分为两部分。

当命令行传入参数为-c，后加一个数字n，则生成n个互不相同的数独解，输出到同目录的sudoku.txt中。

当命令行传入参数为-s，后加一个文件d（同时支持绝对路径与相对路径），则从d中读取若干个未完成的数独，分别求出它们的任意一个可行解，输出到同目录的sudoku.txt中。

异常处理：遇到以下行为之一，应当抛出错误，并退出程序。

* sudoku -c 后无输入或不是合法的数字输入。错误编号：233。
* sudoku -s 后无输入或不是合法的文件输入。错误编号：2333。
* sudoku -s d 在文件d中有一个无解的待求解数独。错误编号：666。
* sudoku -s d 在文件d中没有以一个完整的数独输入结尾，即不合法输入。错误编号：23333。

1. 性能需求

测试数据满足：

* 当输入为sudoku -c n 时，保证。
* 当输入为sudoku -s d 时，保证文件d中的待求解数独数不超过。

时间限制：一分钟。空间限制：512MB。

1. 结构化需求分析与建模

* 数据建模

涉及到的实体有：

待求解的数独。

合法的数独解。

1 1

合法的数独解

求解任意一组解

待求解的数独

变换

1 1

* 功能建模

文件输出

用户输入

数独求解

-s

用户输入

操作分流

用户输入

-c

数独变换

预设的数独解

* 行为建模
* -s

数独求解

读入数据

分析参数

-c

文件输出

数独变换

读取预设数独

## 具体设计

一、主模块：实现程序的基本功能。

1. init()函数

介绍：为实现功能，程序需要一个初始化函数，用于处理和检测输入、判定操作类型等。

原型：int init(int argc,char\*\* argv);

参数说明：把主函数的参数原样传入，以处理输入。

返回值说明：返回值为0，代表执行一个’-c’操作。返回值为1，代表执行一个’-s’操作。

1. work\_c()函数

介绍：处理’-c’操作的代码部分，用函数封装，而不用在主函数中堆砌。

原型：void work\_c(int n);

参数说明：传入数据是一个数字n，表示要生成的数独个数。

返回值说明：无返回值(void)。

1. work\_s()函数

介绍：处理’-s’操作的代码部分，用函数封装，而不用在主函数中堆砌。

原型：void work\_s();

参数说明：无输入。由于输入输出改版，文件路径不需要再作为参数传入。

返回值说明：无返回值(void)。

1. class Sudoku 数独类型

介绍：用于存储单个数独信息的结构体。

原型：

class Sudoku{

public:

int d[9][9];

Sudoku(){};

void init();

void out1();

void out2();

};

成员说明：

public int d[9][9]; 一个存储数独内数字的二维矩阵，取值范围为0~9。1~9表示已经填入对应数字，0表示未确定此处的数字

public Sudoku(){} 构造函数，数独创建时自动把一个已经填入数字的样例数独赋值给它

public void init(); 初始化函数（从文件中输入）。用于从文件中读取给定的数独局面。由于输入输出改版，文件路径不需要作为参数传入

public void out\_s(); 输出函数，适用于’-s’操作的输出函数

public void out\_c(); 输出函数，适用于‘-c’操作的输出函数

二、DLX模块：基于循环十字链表的精确覆盖问题算法，用于实现数独求解。

介绍：实现数独残局求解的模块。

原型：

成员说明：

class DLX{

private:

const static int maxn=740;

const static int maxm=330;

const static int maxnode=200010;

int L[maxnode],R[maxnode],U[maxnode],D[maxnode];

int size,col[maxnode],row[maxnode],s[maxm],H[maxn];

bool vis[maxm];

void del(int c);

void add(int c);

public:

int ans[maxn],cnt;

void init(int n,int m);

void link(int r,int c);

bool dfs(int k);

};

private：

const static int maxn=740; 矩阵的最大行数

const static int maxm=330; 矩阵的最大列数

const static int maxnode=200010; 矩阵中的最多’1’个数

int L[maxnode],R[maxnode],U[maxnode],D[maxnode]; 用到的一些变量

int size,col[maxnode],row[maxnode],s[maxm],H[maxn]; 用到的一些变量

bool vis[maxm];用到的一些变量

void add(int c); 尝试选择第c行

void del(int c); 尝试不选择第c行

public:

int ans[maxn],cnt; 用于保存哪几行最终被选中，下标范围为[0,cnt-1]

void init(int n, int m); 初始化n\*m大小的矩阵

void link(int r, int c); 把第r行第c列的位置强行设置为1（用于已知条件）

bool dfs(int k); 执行第k层的深度优先搜索（是递归过程，会自己调用自己）。外层调用时执行 dfs(0); 即可

三、快速输入/输出模块：用于加速程序的I/O。

介绍：比c++自带输出输出函数更快的输入输出实现。

原型：

namespace fastIO {

struct fileIO{

FILE \*\_\_fastIOInFile=NULL,\*\_\_fastIOOutFile=NULL;

~fileIO(){}

}fIO;

bool refreshIn(const char\* file);

bool refreshOut(const char\* file);

//======================fread======================

bool IOerror=0;

static char ibuf[BUF\_SIZE],\*pi=ibuf+BUF\_SIZE-1;

const static char \*iend=ibuf+BUF\_SIZE;

bool blank(const register char &ch);

void read(register int &x);

//======================fwrite=====================

static char obuf[BUF\_SIZE],\*po=obuf;

const static char \*oend=obuf+BUF\_SIZE;

struct Ostream\_fwrite {

void out(const register char &ch);

~Ostream\_fwrite(){}

}Ostream;

#define print(x) Ostream.out(x)

};

成员说明：

struct fileIO 用于定向文件输入输出

FILE \*\_\_fastIOInFile 输入文件的指针

FILE \*\_\_fastIOOutFile 输出文件的指针

~fileIO(){}析构函数，用于在程序结束时自动处理打开或关闭的文件指针

bool refreshIn(const char\* file); 打开传入字符串对应的文件作为输入流，返回值为是否打开成功。

bool refreshOut(const char\* file); 打开传入字符串对应的文件作为输出流，返回值为是否打开成功。

bool IOerror=0; 用于判定输入结束或出现异常

static char ibuf[BUF\_SIZE],\*pi=ibuf+BUF\_SIZE-1; 用于fread输入优化的缓冲区，以及它的头指针

const static char \*iend=ibuf+BUF\_SIZE; 输入缓冲区的尾指针

bool blank(const register char &ch); 判定传入字符是否为空格/制表符等可忽略字符

void read(register int &x); 从缓冲区里加载下一个数字，并赋值给x

static char obuf[BUF\_SIZE],\*po=obuf; 用于fwrite输出优化的缓冲区，以及它的头指针

const static char \*oend=obuf+BUF\_SIZE; 输出缓冲区的尾指针

struct Ostream\_fwrite 实现缓冲区满自动写出

void out(const register char &ch); 在缓冲区中加载下一个字符为ch

~Ostream\_fwrite(){} 析构函数，用于在程序结束时自动把缓冲区剩余部分写出