

专题一：结构化程序设计与递归函数

实验报告

报告小组：47 | 小组成员：郭书廷(3170104871),叶帆(3170102410) | 指导老师：张引，田沈晶，熊海辉

项目：数字化分，走迷宫，数独游戏

Project1-数字划分

问题描述

编写递归程序，求正整数的所有不同的划分组合。

算法分析与
代码技巧

方法一：
每一层递归获得两个参数，S，last，表示当前拆分的数字是S，最小的拆分数字必须不小于last。此方法按方案的字典序输出。
则令 F(S,last)表示数字 S 的拆分方案，那么
F(S,last)=for_each(last<=i<=n){i+F(n-i,i)}
最终答案为 F(n,1)

方法二：
在方法一的基础上更改搜索顺序，与方法一的输出顺序相反。
F(S,last)=for_each(1<=i<=min(S,last)){i+F(n-i,i)}

方法三：
在方法一的基础上增加深度参数 depth 和深度限制 Lim，当 depth==Lim 或 S<=0 时返回。则可以按 拆分方案中的数字个数 顺序输出。

方案的记录：
使用一个数组模拟栈的工作原理，每次向栈压入一个数字，最后，顺序输出栈中的数字即为拆分方案。具体如下：
压入元素 x: Stack[++top]=x;
弹出元素: Stack[top--]=0;

代码结构及
函数解释

main()

partition_x

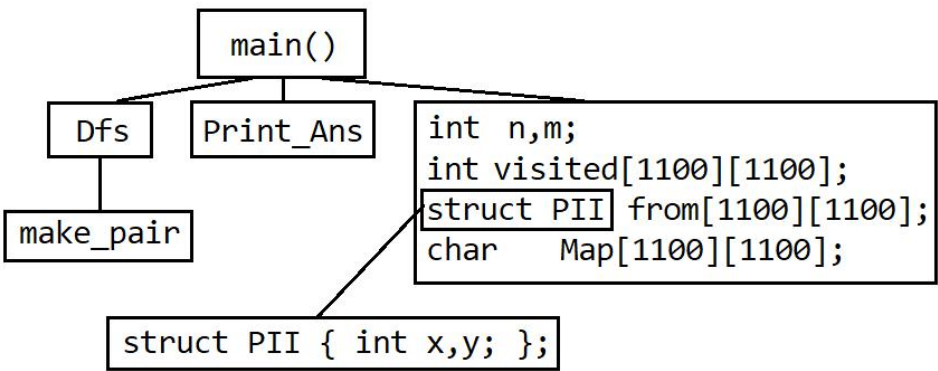
Dfs

Print

int L[1100],top;

```
void partition_x(const int x); //计算方案的接口
void Dfs(...); //主搜索函数
void Print(const int N); //方案输出函数，当前输出的数字总和为 N。
int L[1100],top; //栈数组和计数器。
```

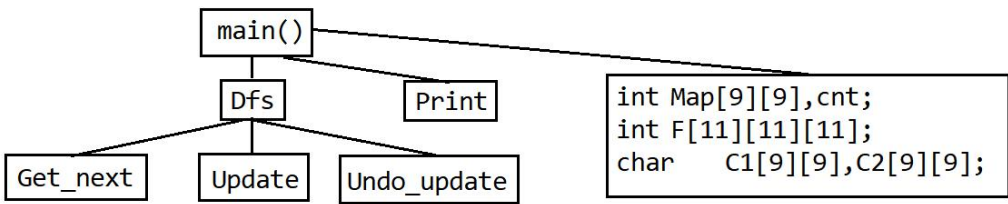
使用方法	打开 Number.exe（方法二为 Number1.exe,方法三 Number2.exe）,输入整数即可，支持多组数据。
Project2-走迷宫	
问题描述	<p>给定一个 $n*m$ 的地图，标定起点和终点，求从起点到终点的路径。</p> <p>输入格式：第一行包含两个整数 n,m 满足 $n,m \leq 500$；接下来的 n 行，每行一个字符串，包含 m 个字符，描述一个迷宫地图，用“#”表示障碍物，“.”表示空地，“S”表示起点，“T”表示终点。</p> <p>输出格式：一个地图，描述所求路径。</p>
算法分析与 代码技巧	<p>本题目要求输入一个迷宫地图，输出从起点到终点的路线。</p> <p>基本思路是从起点(S_x, S_y)每次枚举该格子上下左右四个方向，直到走到终点(T_x, T_y)。</p> <p>方法一： 如果使用递归方法，则可以使用深度优先搜索算法，但此方法不能保证答案步数最优。</p> <p>方法二： 如果要求答案步数最少，则使用广度优先搜索算法，但此方法通常不使用递归函数实现。</p> <p>代码技巧：</p> <p>1.考虑下面两种写法，使用第二种写法有助于缩短代码长度。</p> <pre>if(x-1,y没有越界 && Map[x-1][y] 没有走过 && Map[x-1][y] 不是障碍物) 继续搜索Map[x-1][y]; if(x+1,y没有越界 && Map[x+1][y] 没有走过 && Map[x+1][y] 不是障碍物) 继续搜索Map[x+1][y]; if(x,y-1没有越界 && Map[x][y-1] 没有走过 && Map[x][y-1] 不是障碍物) 继续搜索Map[x][y-1]; if(x,y+1没有越界 && Map[x][y+1] 没有走过 && Map[x][y+1] 不是障碍物) 继续搜索Map[x][y+1]; int dx[4]={-1,0,1,0},dy[4]={0,1,0,-1}; for(i=0;i<4;++i) { int tx=x+dx[i],ty=y+dy[i]; if(tx,ty没有越界 && Map[tx][ty] 没有走过 && Map[tx][ty] 不是障碍物) 继续搜索Map[tx][ty]; }</pre> <p>2.使用文件读入和输出方法，方便操作。默认读入文件名为 in.txt，答案文件名为 out.txt。</p>

程序结构（方法一）	 <pre>graph TD main["main()"] --> Dfs["Dfs"] main --> Print_Ans["Print_Ans"] main --> vars["int n,m; int visited[1100][1100]; struct PII from[1100][1100]; char Map[1100][1100];"] Dfs --> make_pair["make_pair"] struct_def["struct PII { int x,y; };"]</pre>
函数、变量说明（方法一）	<pre>struct PII { int x,y; }; //pair_int_int 包含两个整数的结构体，用来方便存储坐标 struct PII make_pair(const int x,const int y); //将两个整数打包成一个 PII int Dfs(const int Sx,const int Sy, const int Tx,const int Ty); //主搜索函数 void Print_Ans(const int Sx,const int Sy,const int Tx,const int Ty); //答案输出函数 int n,m; //地图尺寸 n*m int visited[1100][1100]; //用于标记[i][j]是否被搜索过 struct PII from[1100][1100]; //记录路径，表示路径中，当前位置的前一个位置 const int dx[4]={-1,0,1,0},dy[4]={0,1,0,-1}; //位移量数组 char Map[1100][1100]; //地图数组</pre>
使用说明（方法一）	<p>在 <code>in.txt</code> 中按约定格式输入地图，运行 <code>Castle_Dfs_like_a_fly.exe</code>（由于 <code>Dfs</code> 方法程序不一定会选择最短路，所以路径看起来像是一只苍蝇乱飞），即可得到答案 <code>out.txt</code>。</p> <p>另外可以双击 <code>gen.exe</code> 输入 <code>n,m</code> 即地图尺寸，（不得大于 <code>500*500</code>），即可随机生成地图至 <code>in.txt</code>，该地图保证从起点到终点有唯一路径。</p> <p>注：查看答案文件时推荐使用 windows 自带的记事本，字体选择黑体，以便获得更好的视觉体验。当系统不能正常显示路径时，可用 <code>prnt_route_temp.h</code> 替换 <code>prnt_route.h</code>，重新编译 <code>Castle_Dfs_like_a_fly.c</code> 运行即可。</p>

程序结构（方法二）	<pre>graph TD main["main()"] --> Bfs["Bfs"] main --> Print_Ans["Print_Ans"] Bfs --> make_pair["make_pair"] subgraph Declarations n["int n,m;"] visited["int visited[1100][1100];"] from["struct PII from[1100][1100];"] Map["char Map[1100][1100];"] end subgraph Definition struct_def["struct PII { int x,y; };"] end struct_def --- from</pre>
函数、变量说明（方法二）	<pre>struct PII { int x,y; }; //pair_int_int 包含两个整数的结构体，用来方便存储坐标 struct PII make_pair(const int x,const int y); //将两个整数打包成一个 PII int Bfs(const int Sx,const int Sy, const int Tx,const int Ty); //主搜索函数 void Print_Ans(const int Sx,const int Sy,const int Tx,const int Ty); //答案输出函数 int n,m; //地图尺寸 n*m int visited[1100][1100]; //用于标记[i][j]是否被搜索过 struct PII from[1100][1100]; //记录路径，表示路径中，当前位置的前一个位置 const int dx[4]={-1,0,1,0},dy[4]={0,1,0,-1}; //位移量数组 char Map[1100][1100]; //地图数组</pre>
使用说明（方法二）	<p>在 in.txt 中按约定格式输入地图，运行 Castle.exe，即可得到答案 out.txt。</p> <p><u>另外可以双击 gen.exe 输入 n,m 即地图尺寸，(不得大于 500*500)，即可随机生成地图至 in.txt，该地图保证从起点到终点有唯一路径。</u></p> <p>注：查看答案文件时推荐使用 windows 自带的记事本，字体选择黑体，以便获得更好的视觉体验。当系统不能正常显示路径时，可用 prnt_route_temp.h 替换 prnt_route.h，重新编译 Castle.c 运行即可。</p>

其他说明	<p>本题代码包含两个.c文件和两个.h文件，程序默认包含 prnt_route.h，当用户无法正常查看地图时可以使用 prnt_route_temp.h 替换该文件，程序将使用标准英文半角符号输出地图。prnt_route.h 中包含答案输出函数，通过 extern 与主程序共用全局变量。</p> <p><u>另外，为了方便查看随机生成的地图，我们提供了 Exchange_Map.exe 会把 in.txt 中的迷宫重新输出至 view.txt，以便查看。</u></p> <p><u>gen.cpp 是由本组成员用 C++写成的地图生成器，需使用如下编译指令进行编译：</u></p> <p><u>g++ gen.cpp -o gen -O2 -std=c++11</u></p> <p>在 gen.cpp 中，有两个参数可以修改，RATE_KEEP_DIR 表示随机生成路径是，保留上一步方向的概率，此参数越大，答案的曲折程度越小，表示格式为千分比，不得等于 1000；RETURN_RATE 表示在一条岔路中不继续搜索而直接返回的概率，此参数可以控制岔路的数量和长度，格式为千分比，不推荐大于 700，否则地图可能不符合期望。</p> <p>另外，有一个 TWIST_RATE 参数，该参数表示每条岔路的最大长度为 $\min(n,m)/TWIST_RATE$，不推荐修改。</p> <p>三个参数的推荐值如下：</p> <pre>const int RATE_KEEP_DIR=120; const int TWIST_RATE=10; const int RETURN_RATE=100;</pre>
Project3-数独游戏	
问题描述	给定 9*9 标准数独题目，求解。

<p>算法分析与 代码技巧</p>	<p>为解决这一问题，我们也尝试了两种方法，准确的说，是第一种方法太慢了，我们对它进行了优化。</p> <p>方法一：用最朴素的方法逐一枚举每一个格子，在某一个格子不能填入任何数字时回溯。</p> <p>这个方法写法相对简单，但对于一些难以求解的情况，程序会非常慢，最慢的甚至无法在 3 秒内得出答案。如果每一个情况需要三秒钟来求解，那么要批量求解数独可能需要等好几分钟。尽管我们将编译器的优化选项开到 O2，一些情况仍需要 1 秒左右。</p> <p>方法二：模拟手算，当给定一道题目时，首先确定每个格子可以填那些数字，每次优先选择可选数字最少的格子，不能填入时回溯。这种方法，可以在填入一个数字后，立即确定数独中有哪些方格已经可以确定，或已经有方格不能填入，实际上是方法一的搜索剪枝。此方法对于大多数数据能在 0.1 秒内求解。（具体样例详见 ppt）</p> <p>代码技巧：</p> <ol style="list-style-type: none">1. 让格子的编号从 0 开始有利于后续处理。2. 如果一个小格子的坐标是 [x][y]，那么它所在的小方格应该是第 $x/3*3+y/3$ 个。3. 如果当前的格子是 [x][y]，那么下一个小格子应该是 $[x+(y+1)/9] [(y+1)\%9]$4. 遍历 3*3 方格 <code>for(i=0;i<9;++i) F[x/3*3+i/3] [y/3*3+i%3] [data] = ...;</code> <p>UI 设计：</p> <ol style="list-style-type: none">1. 用户可以选择两种方式读入：终端手工输入或者文件读入。用户可以输入文件名，程序会自动读至文件尾，并生成答案文件 Result.txt2. 手工输入时程序会给出提示符，用文件读入时，使用数字 0~9 的 9*9 方阵，0 表示空位。3. 为了防止用户输入的数独的解过多，程序对于每个数独最多只输出前 5000 个解。4. 对于不合法的初始状态予以检测，并对错误题目返回错误信息。5. 对于无解数独，输出反馈信息。6. 用户可以看到程序计算每个数独所花费的时间以毫秒为单位。7. 输出的结果用字符绘制表格，以便查看，并标注已知方格。
<p>程序结构（方法一）</p>	<div><div><div>main()</div><div>Dfs</div><div>Print</div></div><div><pre>/*Sudoku save in array Map, index counts from 0.*/ int Map[9][9],cnt; /*R<->Row B<->Block C<->Column*/ int R[11][11],B[11][11],C[11][11]; char C1[9][9],C2[9][9];</pre></div></div>

<p>函数、变量说明（方法一）</p>	<pre>int Map[9][9],cnt; //Map 用于记录当前棋盘，cnt 用于统计已找到的答案个数， 当答案个数超过 5000 时停止搜索。 int R[11][11],B[11][11],C[11][11]; // 分别表示第 i 行/块/列的数字 j 是否已 被用过。 char C1[9][9],C2[9][9]; //用于标记已知方格 int Dfs(const int x,const int y); //主搜索函数，表示当前搜索方格[x][y]需 填入的数字。 void Print(int Map[9][9],char C1[9][9],char C2[9][9]); //答案输出函 数，C1 和 C2 分别是每个数字左侧和右边的符号，对于已知的数字，会用[]标记，其他 数字两侧输出空格。</pre>
<p>程序结构（方法二）</p>	 <pre>graph TD main["main()"] --> Dfs["Dfs"] main --> Print["Print"] main --> vars["int Map[9][9],cnt; int F[11][11][11]; char C1[9][9],C2[9][9];"] Dfs --> Get_next["Get_next"] Dfs --> Update["Update"] Dfs --> Undo_update["Undo_update"]</pre>
<p>函数、变量说明（方法二）</p>	<pre>int Map[9][9],cnt; //Map 用于记录当前棋盘，cnt 用于统计已找到的答案个数， 当答案个数超过 5000 时停止搜索。 int F[11][11][11]; //F[i][j][k]表示方格[i][j]的数字 k 是否可用。 char C1[9][9],C2[9][9]; //用于标记已知方格 struct PII { int x,y; }; //pair_int_int 包含两个整数的结构体，用来方便存 储坐标 int Dfs(const int rest); //主搜索函数，表示还有 rest 个方格需要填入。 struct PII Get_next(); //返回当前棋盘中可用数字最少的方格，如果存在方格未 填入数字，且无法填入数字，那么返回值的 x 坐标为-1。 void Update(const int x,const int y,const int data); //表示在[x][y] 填入 data，仅更新数组 F。 void Undo_update(const int x,const int y,const int data); //表示撤销 对[x][y]填入 data 的操作，仅更新数组 F void Print(int Map[9][9],char C1[9][9],char C2[9][9]); //答案输出函 数，C1 和 C2 分别是每个数字左侧和右边的符号，对于已知的数字，会用[]标记，其他 数字两侧输出空格。</pre>

使用说明	<p>点击 Sudoku.exe (方法二使用 Sudoku1.exe)，选择手工键入(1)或文件读入(2)；键盘输入模式输入 9*9 包含 0~9 的方阵即可，文件模式需输入文件名。</p> <p>程序支持多组测试数据，文件读入时会自动读到文件尾并将但输出至 Result.txt，win 控制台若要结束程序可以按 Ctrl+Z 在按回车，Linux 终端按 Ctrl+d。</p> <p>注：本程序的警告信息会通过 stderr 输出，不会输出到文件。</p>
其他说明	<p>答案输出函数位于 prnt_sudoku.h 中，请勿删除。</p>