Assignment_4　N皇后问题

一、项目简介：

八皇后问题是一个古老而著名的问题，是回溯算法的经典问题。该问题是十九世纪著名的数学家高斯在1850年提出的：在8*8的国际象棋棋盘上，安放8个皇后，要求没有一个皇后能够“吃掉”任何其它一个皇后，即任意两个皇后不能处于同一行，同一列或者同一条对角线上，求解有多少种摆法。

高斯认为有76种方案。1854年在柏林的象棋杂志上不同的作者发表了40种不同的解，后来有人用图论的方法得到结论，有92种摆法。

本实验拓展了N皇后问题，即皇后个数由用户输入。

八皇后在棋盘上分布的各种可能的格局数目非常大，约等于2的32次方种，但是，可以将一些明显不满足问题要求的格局排除掉。由于任意两个皇后不能同行，即每行只能放置一个皇后，因此将第i个皇后放在第i行上，这样在放置第i个皇后时，只要考虑它与前i-1个皇后处于不同列和不同对角线位置上即可。

二、构建设想：

解决这个问题采用回溯法，首先将第一个皇后放置在第一行第一列，然后，依次在下一行上放置一个皇后，直到八个皇后全部放置安全。在放置每个皇后时，都依次对每一列进行检测，首先检测放在第一列是否与已放置的皇后冲突，如不冲突，则将皇后放置在该列，否则，选择改行的下一列进行检测。如整行的八列都冲突，则回到上一行，重新选择位置，依次类推。

三、程序设计：

1. 程序的类与结构组织

整个程序都是在Queens类中进行,包括棋盘的初始化，主要的递归函数，以及打印棋盘的功能，如下是Queens类中变量与函数的声明:

```
class Queens{

private:
    int      _N;                          //皇后的数量
    int      _count;                      //解法的数量
    char**   _chessboard;                 //棋盘
    char*    _queensPos;                  //皇后的位置

public:

    void print();                         //打印棋盘
    void initChess(int N);                //初始化棋盘   包括初始化皇后的位置数组
    void findQueens(int i);               //主递归程序，寻找N皇后的解法
    int  getCount(){ return _count; }     //返回N皇后的解法
    bool isPutable(int i,int j);          //判断如果把皇后放在这个位置是不是可以接受
的

};
```

2.Queens类中函数的实现

(1) 初始化函数 initChess() :

初始化函数的主要作用是为N皇后所需要的棋盘申请空间,以及对棋盘的每个位置赋上初始值,如下是initChess()函数的具体实现:

```
//初始化棋盘   包括初始化皇后的位置数组
void Queens::initChess(int N)
{
    _N = N;
    _count = 0;
    _queensPos = new char[N];
    _chessboard = new char*[N];

    for(int i = 0; i < N; i++)
        _chessboard[i] = new char[N];

    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            _chessboard[i][j] = '0';
}
```

(2)主递归函数 findQueens() :

递归函数的主要思想是对下一行的每一个空位进行皇后位置的赋值，如果检测到是可以放置的就进入下一层,否则就返回上层，知道得到所有的赋值，如下是findQueens()函数的具体实现:

```
//主递归程序，寻找N皇后的解法
void Queens::findQueens(int i)
{
    if(i == _N)
    {
        print();
        _count++;
        cout << "\n\n";
        return ;
    }
    else
    {
        for(int k = 0; k < N; k++)
        {
            _chessboard[i][k] = 'X';
            _queensPos[i] = k;
            if(isPutable(i, k))
                findQueens(i + 1);
            _chessboard[i][k] = '0';
        }
    }
}
```

(3) 判断函数 isPutable() :

每放置一个皇后之前都要判断是否满足N皇后的规则，如果没有冲突的话就return true否则 return false，如下是函数的具体实现方式:

```
//判断是否可以放置皇后
bool Queens::isPutable(int i, int j)
{
    for(int k = 0; k < i; k++)
        if(j == _queensPos[k] || abs(i - k) == abs( j - _queensPos[k]))
            return false;

    return true;
}
```

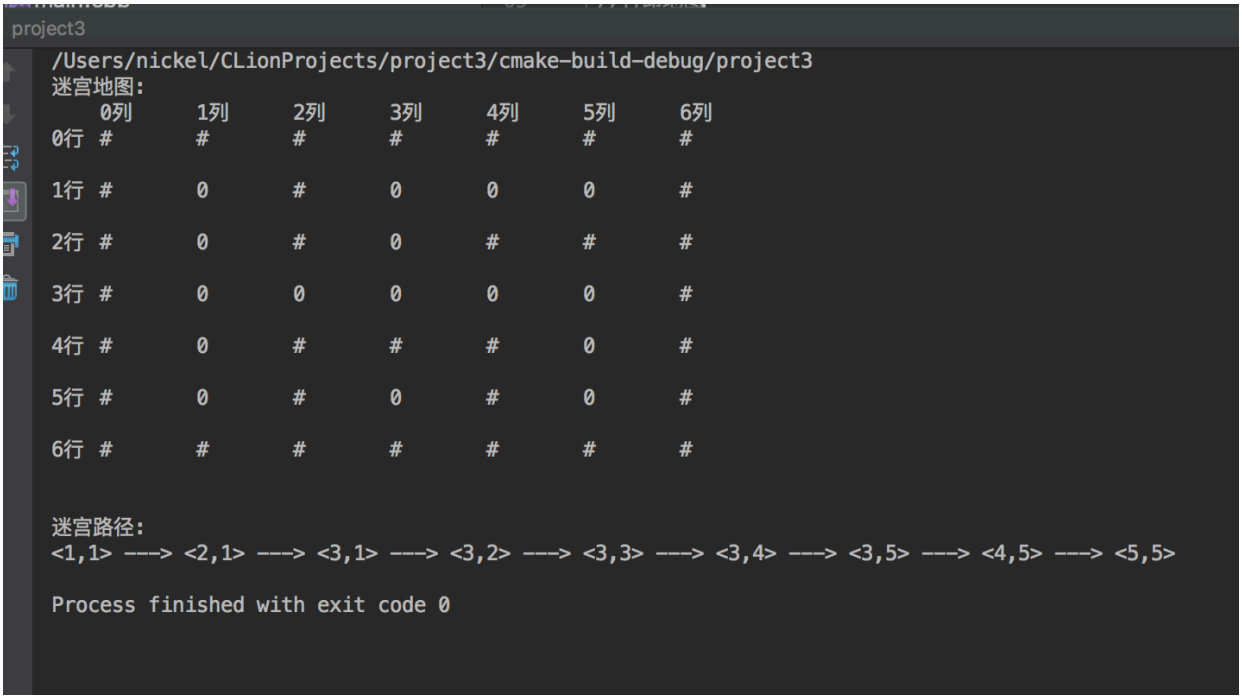
(4) 打印函数 print():

每找到一种解法都要打印一次棋盘，如下是打印函数的具体实现方式:

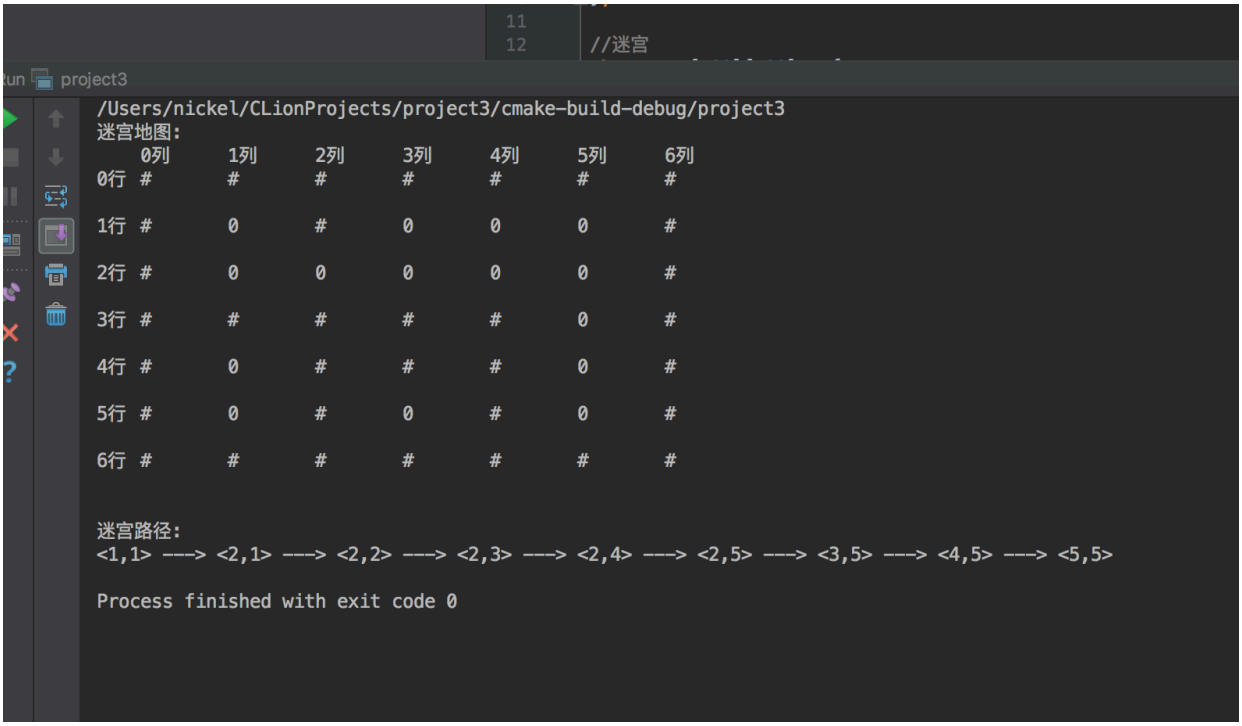
```
//打印棋盘
void Queens::print()
{
    for(int i = 0; i < _N; i++)
    {
        for (int j = 0; j < _N; j++)
            cout << _chessboard[i][j] << "    ";

        cout << endl;
    }
}
```

四、N皇后运行截图：

1. 皇后数量为6时：



2.皇后数量为8时：



3.皇后数量为10时:

4.皇后数量为14时:

以及:当皇后数量为14时,解法的数量已经变成了365596种,我得到最后结果时已经花费了一段不短的时间,所以当皇后数量>=15时由于计算量过大就不在赘述~

五、程序容错性测试:

1.程序的容错性测试1：

当皇后数量 == 3时:

2.程序的容错性测试2：

当皇后数量 == 2时:

3.程序的容错性测试3：

当皇后数量 == 0时: