

**Lab Report**

**实验报告**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2024-2025** |
| **Major**: | Software Engineering |
| **Class**: | 2023 |
| **Student Name**: | 吴仲雄 |
| **Student ID:** | 222023321062007 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Recursion and Backtracking Framework  递归和回溯框架 | | | |
| Date | | Oct，2024 | Type | | ☑Confirmatory （验证确认型）  ☑Design（设计型）  🗆Comprehensive（综合型） |
| 1. **Objective & Requirements（实验目的）**    1. Understand the design and implementation principles of recursion and backtracking algorithm to solve a practical problem   理解递归和回溯算法的原理和实现方法   * 1. Understand the object-oriented design of backtracking framework using C++, especially the iterator inner class   理解采用面向对象思想设计通用回溯算法框架的思路，尤其是其中基于迭代器的通用算法的设计思想   * 1. Grasp the use of the backtracking framework to solve a specific problem, i.e. the 8-Queen problem.   掌握面向对象回溯框架的使用，以 8皇后问题为例培养利用回溯框架解决实际问题的能力 | | | | | |
| 1. **Experimental environment (**platform and software**)（实验环境）**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. Experimental content and design (Main Content, Procedure, Codes and Results)（此部分应包含每一个实验内容的详细设计，含实验思路、详细实验步骤、核心代码说明等）   Task 1  A chessboard has eight rows and eight columns. In the game of chess, the queen is the most powerful piece: she can attack any piece in her row, any piece in her column, and any piece in either of her diagonals. Develop and validate a program to place eight queens on a chessboard in such a way that no queen is under attack from any other queen.    在8行乘8列的棋盘上，放置8枚皇后棋子，使得任意两个皇后棋子不可互相攻击，即不在同一行、同一列和同一对角线上。如上图所示。  Requirement（要求）:   1. You should use the backtrack framework to solve the 8-Queen problem.   使用所提供通用回溯框架解决该问题。   1. The codes in 4 of the files are fixed and you are not allowed to modify them 2. main.cpp 3. backtrack.h 4. backtrack.cpp 5. application.h   以上4个文件中的代码不允许做任何修改。   1. The codes in 3 of the files are to be implemented： 2. position.h 3. position.cpp 4. application.cpp (including the iterator)   以上3个文件中的代码需要根据实际问题进行实现.  position.h:  #ifndef POSITION\_H  #define POSITION\_H  class Position  {  public:  Position();  Position(int thisRow, int thisCol);  int getRow() const;  int getColumn() const;  private:  int row;  int column;  };  #endif  position.cpp:  #include "position.h"  Position::Position()  {  row = 0;  column = 0;  }  Position::Position(int thisRow, int thisCol)  {  row = thisRow;  column = thisCol;  }  int Position::getRow() const  {  return row;  }  int Position::getColumn() const  {  return column;  }  application.cpp:  #include "application.h"  #include <iostream>  #include <cmath>  using namespace std;  const int BOARD\_SIZE = 8;  static int solution[BOARD\_SIZE]; // solution[i] = 皇后在第i行的列位置  static int numQueens; // 已放置的皇后数量  void Application::initialize()  {  numQueens = 0;  for (int i = 0; i < BOARD\_SIZE; i++)  solution[i] = -1; // 初始化为-1  }  Position Application::getStartPosition()  {  initialize();  // 起始位置行号为 -1  Position startPosition(-1, -1);  return startPosition;  }  bool Application::isValid(const Position& p)  {  int row = p.getRow();  int col = p.getColumn();  for (int i = 0; i < row; i++)  {  int existingCol = solution[i];  // 检查列冲突  if (col == existingCol)  return false;  // 检查对角线冲突  if (abs(row - i) == abs(col - existingCol))  return false;  }  return true;  }  void Application::progress(const Position& p)  {  int row = p.getRow();  int col = p.getColumn();  if (row >= 0)  {  solution[row] = col;  numQueens++;  }  }  bool Application::success(const Position& p)  {  return numQueens == BOARD\_SIZE;  }  void Application::goBack(const Position& p)  {  int row = p.getRow();  if (row >= 0)  {  solution[row] = -1;  numQueens--;  }  }  void Application::print()  {  cout << "One solution to this problem is:" << endl;  for (int i = 0; i < BOARD\_SIZE; i++)  {  cout << "|";  for (int j = 0; j < BOARD\_SIZE; j++)  {  if (solution[i] == j)  cout << "Q|";  else  cout << "\_|";  }  cout << endl;  }  }  // Iterator 实现  // 定义用于存储迭代器状态的结构体  struct IteratorState  {  int row; // 当前行号  int col; // 当前列号  };  Application::Iterator::Iterator()  {  currItrPosPtr = nullptr;  }  Application::Iterator::Iterator(const Position& currP)  {  // 分配迭代器状态并初始化  IteratorState\* state = new IteratorState;  state->row = currP.getRow() + 1; // 移动到下一行  state->col = 0; // 从第0列开始  if (state->row < BOARD\_SIZE)  {  currItrPosPtr = state;  }  else  {  delete state;  currItrPosPtr = nullptr;  }  }  Position Application::Iterator::getNextPosition()  {  IteratorState\* state = (IteratorState\*)currItrPosPtr;  Position nextPos(state->row, state->col);  state->col++; // 为下次调用递增列号  return nextPos;  }  bool Application::Iterator::noNextPosition()  {  IteratorState\* state = (IteratorState\*)currItrPosPtr;  if (state == nullptr)  return true;  if (state->col >= BOARD\_SIZE)  return true;  else  return false;  }  Application::Iterator::~Iterator()  {  if (currItrPosPtr != nullptr)  {  delete (IteratorState\*)currItrPosPtr;  currItrPosPtr = nullptr;  }  }  实现结果如下： | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）此部分应包含实验结果，对实验结果的分析，实验收获的总结，实验中存在问题的讨论等；另外，需要回应一下如下思考题：   1. 在Application的内部类Iterator中为什么封装了一个无类型指针，其作用是什么？  2. 如何修改使你的程序以找出这个棋盘上所有可能的八个皇后放置位置？（可以概述基本思路，也可以修改并运行代码得到准确结果。）   1. 在 Application的内部类Iterator中封装一个无类型指针currItrPosPtr的主要目的是为了实现迭代器的状态存储和管理。无类型指针(void\*)可以指向任何类型的数据，这使得它非常灵活，可以在运行时动态地分配和管理不同类型的数据结构。   这种设计的好处包括：  (1) 封装性：将迭代器的状态信息封装在一个结构体中，并通过无类型指针进行访问，可以隐藏实现细节，提高代码的封装性。  (2) 灵活性：无类型指针可以指向任何类型的数据，使得代码更加灵活，可以在运行时动态地分配和管理不同类型的数据结构。  (3) 内存管理：通过动态分配和释放内存，可以更好地管理迭代器的生命周期，避免内存泄漏。  2.实现代码如下(修改了backtrack.cpp与application.cpp)：  backtrack.cpp:  #include "backtrack.h"  BackTrack::BackTrack() {}  BackTrack::BackTrack(const Application& thisApp)  {  this->app = thisApp;  }  bool BackTrack::backTrack(const Position& currPos)  {  app.progress(currPos);  if (app.success(currPos))  {  app.print(); // 打印当前解  app.goBack(currPos); // 回溯以寻找下一个解  return false; // 继续寻找其他解  }  else  {  Application::Iterator itr(currPos);  Position nextPos;  while (!itr.noNextPosition())  {  nextPos = itr.getNextPosition();  if (app.isValid(nextPos) && backTrack(nextPos)) // 递归调用  return true;  }  // 无法从 currPos 到达目标  app.goBack(currPos); // 回溯  return false; // 从 currPos 无解  }  }  application.cpp:  #include "application.h"  #include <iostream>  #include <cmath>  using namespace std;  const int BOARD\_SIZE = 8;  static int solution[BOARD\_SIZE]; // solution[i] = 皇后在第i行的列位置  static int numQueens; // 已放置的皇后数量  static int solutionCount; // 解的数量  void Application::initialize()  {  numQueens = 0;  solutionCount = 0;  for (int i = 0; i < BOARD\_SIZE; i++)  solution[i] = -1; // 初始化为-1  }  Position Application::getStartPosition()  {  initialize();  // 起始位置行号为 -1  Position startPosition(-1, -1);  return startPosition;  }  bool Application::isValid(const Position& p)  {  int row = p.getRow();  int col = p.getColumn();  for (int i = 0; i < row; i++)  {  int existingCol = solution[i];  // 检查列冲突  if (col == existingCol)  return false;  // 检查对角线冲突  if (abs(row - i) == abs(col - existingCol))  return false;  }  return true;  }  void Application::progress(const Position& p)  {  int row = p.getRow();  int col = p.getColumn();  if (row >= 0)  {  solution[row] = col;  numQueens++;  }  }  bool Application::success(const Position& p)  {  return numQueens == BOARD\_SIZE;  }  void Application::goBack(const Position& p)  {  int row = p.getRow();  if (row >= 0)  {  solution[row] = -1;  numQueens--;  }  }  void Application::print()  {  solutionCount++;  cout << "Solution #" << solutionCount << ":" << endl;  for (int i = 0; i < BOARD\_SIZE; i++)  {  cout << "|";  for (int j = 0; j < BOARD\_SIZE; j++)  {  if (solution[i] == j)  cout << "Q|";  else  cout << "\_|";  }  cout << endl;  }  cout << endl;  }  // Iterator 实现  // 定义用于存储迭代器状态的结构体  struct IteratorState  {  int row; // 当前行号  int col; // 当前列号  };  Application::Iterator::Iterator()  {  currItrPosPtr = nullptr;  }  Application::Iterator::Iterator(const Position& currP)  {  // 分配迭代器状态并初始化  IteratorState\* state = new IteratorState;  state->row = currP.getRow() + 1; // 移动到下一行  state->col = 0; // 从第0列开始  if (state->row < BOARD\_SIZE)  {  currItrPosPtr = state;  }  else  {  delete state;  currItrPosPtr = nullptr;  }  }  Position Application::Iterator::getNextPosition()  {  IteratorState\* state = (IteratorState\*)currItrPosPtr;  Position nextPos(state->row, state->col);  state->col++; // 为下次调用递增列号  return nextPos;  }  bool Application::Iterator::noNextPosition()  {  IteratorState\* state = (IteratorState\*)currItrPosPtr;  if (state == nullptr)  return true;  if (state->col >= BOARD\_SIZE)  return true;  else  return false;  }  Application::Iterator::~Iterator()  {  if (currItrPosPtr != nullptr)  {  delete (IteratorState\*)currItrPosPtr;  currItrPosPtr = nullptr;  }  }  实验结果如下： | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |