Abteilung für
Angewandte Mathematik

SoSe 2019
Christian Palus

Einführung in die Programmierung für Studierende der Naturwissenschaften

Blatt 11 (Bonusblatt) – 15.07.2019

Abgabe: Briefkästen RZ/E-Mail bis Montag, den 22.07.2019, 16:00 Uhr

---

Aufgabe 1 (20 Bonuspunkte)**.** Wir wollen ein C++-Programm zum Auffinden einer Lösung des N-Damenproblems (vgl. Blatt 10, Aufgabe 1) schreiben. Dazu identifizieren wir das $N \times N$-Spielbrett mit einem dynamischen Array `int board[N][N]`, wobei eine in einem Feld platzierte Dame durch den Eintrag 1 und ein leeres Feld durch den Eintrag 0 gekennzeichnet wird. Das Problem selbst lässt sich gut durch die folgende Klasse `NQueensProblem` darstellen:

```
class NQueensProblem {
public:
  // Konstruktor: initialisiert leeres Brett mit dynamischem Speicher
  NQueensProblem( int );
  // Destruktor: gibt Speicher wieder frei
  ~NQueensProblem();
  // prueft ob ein Feld angegriffen oder besetzt ist
  bool is_attacked( int, int );
  // loest das Problem rekursiv
  bool solve();
  // gibt aktuellen Zustand des Bretts aus
  void print_state();
private:
  const int N;  // Brettgroesse
  int n_queens; // Anzahl noch zu platzierender Damen
  int** board;  // Brett als Array (leeres Feld=0, besetztes Feld=1)
};
```

Die möglichen Zustände sind durch die privaten Attribute dieser Klasse festgelegt. Der folgende Konstruktor reserviert den nötigen Speicher und initialisiert alle Einträge des Arrays `board` mit 0 sowie die Anzahl `n_queens` der zu platzierenden Damen mit N, der dazugehörende Destruktor gibt den Speicherplatz wieder frei:

```
NQueensProblem::NQueensProblem( int board_size ) : N(board_size), n_queens(board_size)
{
  // erzeugt ein dynamisches Array der Groesse N aus int-Zeigern
  board = new int*[N];
  // erzeugt fuer jeden dieser Zeiger ein int-Array der Groesse N
  for (int i = 0; i < N; ++i)
    board[i] = new int[N];
  // initialisiert alle Eintraege mit 0
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      board[i][j] = 0;
    }
  }
}
```

Fortsetzung auf der Rückseite.

```
NQueensProblem::~NQueensProblem()
{
  for (int i = 0; i < N; ++i) {
    delete[] board[i];
  }
  delete[] board;
}
```

Auf der Vorlesungshomepage finden Sie die Datei `nqueens.cc`, welche die obige Deklaration der Klasse `NQueensProblem` sowie die Definitionen des Konstruktors und Destruktors enthält. Die Definitionen der Methoden `NQueensProblem::is_attacked(int i, int j)`, `NQueensProblem::solve()` sowie `NQueensProblem::print_state()` fehlen noch. Diese sollen die folgenden Aufgaben erledigen:

`is_attacked(i,j)`	Soll **`true`** zurückgeben, falls `board[i][j]==1` oder falls `board[k][l]==1` für ein Feld `(k,l)` in der selben Zeile, Spalte oder Diagonale wie `(i,j)`.
`solve()`	Soll das N-Damenproblem mittels rekursivem Backtracking lösen (vgl. Blatt 10, Aufgabe 2). Dabei kann die Methode `is_attacked()` verwendet werden um festzustellen, ob das Platzieren einer Dame in einem Feld ein zulässiger Teilschritt ist.
`print_state()`	Soll die Brettmatrix `board` in übersichtlicher Weise in der Konsole ausgeben, wobei platzierte Damen durch das Zeichen `Q` gekennzeichnet werden (siehte unten).

Die Ausgabe der Methode `print_state()` sollte für eine Lösung beispielsweise für $N = 5$ die folgende Form haben:

```
---------------------
| Q |   |   |   |   |
---------------------
|   |   | Q |   |   |
---------------------
|   |   |   |   | Q |
---------------------
|   | Q |   |   |   |
---------------------
|   |   |   | Q |   |
---------------------
```

Ergänzen Sie die Datei `nqueens.cc` um die Definitionen der drei Methoden aus der obigen Tabelle. Schreiben Sie außerdem eine `main`-Funktion. Diese soll beim Programmaufruf die Brettgröße N als Argument aus der Kommandozeile übergeben bekommen und eine Instanz der Klasse `NQueensProblem` mit dieser Brettgröße erzeugen. Rufen Sie in der `main`-Funktion anschließend die Methode `solve()` des instanziierten Objekts auf und lassen Sie das Spielbrett im Lösungszustand durch die Methode `print_state()` ausgeben.
Für die Bearbeitung dieser Aufgabe werden 20 Bonuspunkte vergeben: Jeweils 5 Bonuspunkte für die drei Klassenmethoden sowie weitere 5 Bonuspunkte für eine korrekte Implementierung der `main`-Funktion.

»Das Fluchen ist die einzige Sprache, die jeder Programmierer beherrscht.«
Eine erholsame vorlesungsfreie Zeit!