1. Übungsblatt

1. Aufgabe

Lesen Sie eine Reihe von int-Zahlen von einer Datei ein, und geben Sie diese in umgekehrter Reihenfolge wieder aus.
Implementieren Sie das Verfahren so, daß Sie in der ersten Variante ein Feld verwenden und in der zweiten Variante das Problem mit Hilfe einer Rekursion lösen.
Wie erkennen Sie in Ihrem Programm, daß Sie am Dateiende angekommen sind und keine weiteren Zahlen eingelesen werden können?

2. Aufgabe

Die Fibbonacci-Zahlen sind wie folgt definiert:

> Die erste Fibonacci-Zahl ist 1.
> Die zweite Fibonacci-Zahl ist 2.
> Jede weitere ergibt sich aus der Summe der beiden vorherigen.

Geben Sie das entsprechende C-Programm an und testen Sie Ihr Programm mit den folgenden Eingaben: 2, 4, 7, -6.

3. Aufgabe (Praktikum)

Implementieren Sie in C ein Backtracking-Verfahren, das zu einem gegebenen Labyrinth den Weg vom Startpunkt zum Ziel findet.

Gehen Sie bei der Entwicklung und Realisierung des Verfahren systematisch vor.

- Welche Datentypen und Operationen benötigen Sie?
- Wie verfeinern Sie die Aufgabe in Teilschritte?
- Welche Tests müssen Sie durchführen?

Ihr Algorithmus soll auf folgenden Ausgangsdaten arbeiten:

Sie geben die Anzahl der Zeilen und Spalten ein (Sie können davon ausgehen, daß maximal 50 Zeilen und 50 Spalten benötigt werden). In einer Datei ist das Labyrinth wie folgt definiert:

Start ist immer an der Position (0,0), das Ziel ist immer bei (Spaltenanzahl - 1, Zeilenanzahl - 1).
Um die Definition des Labyrinthes so komfortabel und einfach für den Benutzer wie möglich zu machen, soll diese formatfrei erfolgen. D.h. es werden in ausreichender Zahl 0 und 1 angegeben, die durch bel. viele Leerzeichen und Zeilenumbrüche angegeben werden.

0 im Feld(i,j) bedeutet, daß dieses Feld frei ist und in einem möglichen Lösungsweg genutzt werden kann,
1 im Feld(i,j) bedeutet Hindernis und kann nicht überwunden werden.

Ihr Programm soll folgende Eingaben verarbeiten:

Anzahl der Zeilen:	z.B.: 8
Anzahl der Spalten:	z.B.: 8
Name der Labyrinth-Datei:	z.B.: labyrint.def

Die Definition des Labyrinths steht dann in der Datei „`labyrint.def`“. Beim Einlesen können Sie davon ausgehen, dass in der Datei ein fehlerfreies Labyrinth mit entsprechender Zeilen- und Spaltenanzahl gespeichert wurde. „`labyrint.def`“ könnte beispielsweise folgende Daten beinhalten:

```
0  1  1  1  1  1  1  0
0  1  1  1  1  1  1  0
0  0  0  0  0  1  1  1
0  1  1  0  1  1  1  1
0  1  0  0  0  0  1  1
0  1  0  1  1  0  1  1
0  1  1  1  1  0  0  1
0  1  0  1  1  1  0  0
```

Wie bereits gesagt ist der Start ist immer bei (0,0). In diesem Beispiel ist das Ziel bei (7,7). In diesem Beispiel könnte das Ergebnis wie folgt aussehen:

Folgender Weg wurde gefunden:

```
+-------------------------+
| x                       |
| x                       |
| x  x  x  x              |
|          x              |
|          x  x  x        |
|             x           |
|             x  x        |
|                x  x     |
+-------------------------+
```

Ergänzende Hinweise:

1. Ihre Lösungsweg darf keine diagonale Schritte sondern nur horizontale oder vertikale Verbindungen besitzen.
2. Ihr Programm sollte auch funktionieren wenn kein Lösungsweg gefunden werden kann. Das Programm sollte dann einen entsprechenden Hinweis ausgeben.