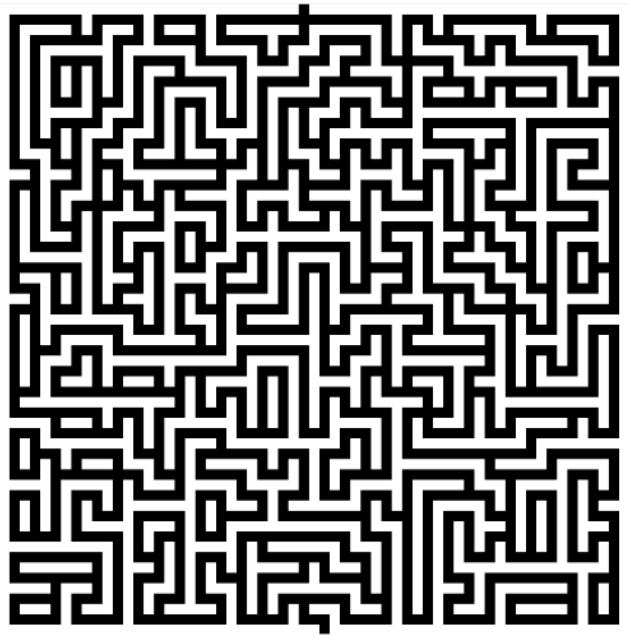
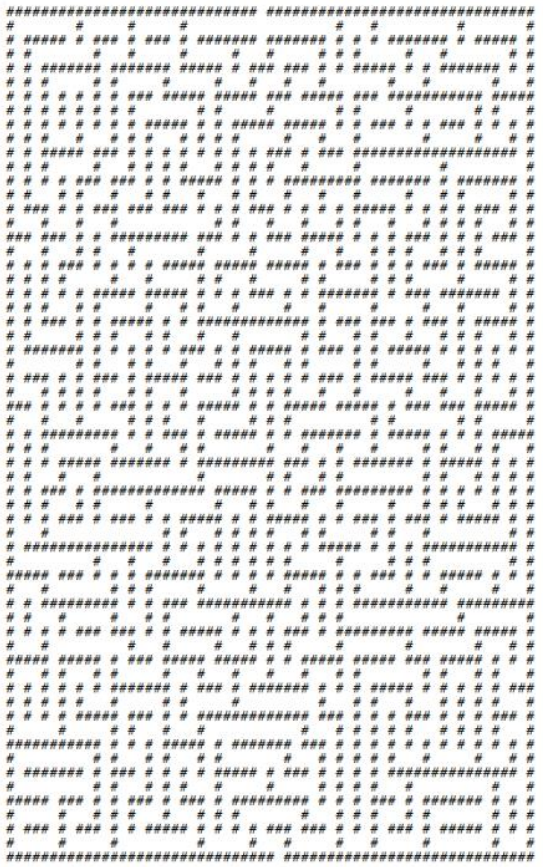
1. Exercise 5: Roboter-Labyrinth

Schreiben Sie ein C++-Programm, das für ein gegebenes Labyrinth einen Weg vom Eingang zum Ausgang findet. Dabei sollen die Klassen und Datenstrukturen designt, das Labyrinth eingelesen, multiple Roboter als Threads implementiert und dann der Weg ausgegeben werden.

2. Darstellung des Labyrinths





Links ist das Labyrinth normal dargestellt, rechts im ASCII-Format der Datei, wie sie von Ihrem Programm eingelesen und dargestellt werden soll (ohne Weg).
Ein Feld wird durch genau ein ASCII-Zeichen dargestellt, dieses stellt entweder eine Mauer (#), ein freies Feld ( ) oder einen Teil des gefundenen Weges (.) dar. Ein Weg hat stets genau die Breite eines einzelnen Felds. Jede Zeile des Labyrinths ist durch einen Newline-Character abgeschlossen. Ein Labyrinth hat stets Rechteckform. Die Größe des Labyrinths soll automatisch bestimmt werden – entscheiden Sie, wie dies in Ihrem Programm erfolgen soll.

Es existiert immer genau ein Eingang und genau ein Ausgang. Das gesamte Labyrinth ist durch eine Mauer begrenzt, der Ein- und Ausgang sind genau die beiden offenen Stellen in der Begrenzungsmauer. Ein- und Ausgang können an beliebigen Stellen der Umgrenzungsmauer sein. Der Eingang ist jene Öffnung, die man zuerst findet, wenn man im Uhrzeigersinn links oben zu suchen beginnt. Die zweite Öffnung ist der Ausgang (Ein- und Ausgang sind nicht speziell markiert). Verzweigungen sind nur horizontal und vertikal möglich, es gibt also höchstens vier Richtungen.

3. Objektorientiertes Design und E/A-Operationen

Implementieren Sie die Kommandozeile und die Ein-/Ausgabeoperationen für das Labyrinth. Das heißt Sie sollen ein Labyrinth einlesen können und es am Bildschirm ausgeben können. Die Kommandozeile soll folgendermaßen aufgebaut sein:

```
labrob DATEINAME [-t1] [-t2] … [-tN] [-h]
```

Der Name des Programms ist `labrob`, der Name der Datei, `DATEINAME`, muss angegeben werden. In dieser Datei ist das Labyrinth im oben beschriebenen ASCII-Format gespeichert. Die Parameter „`-t1` bis `-tN`“ geben die Typen der Roboter an, die verwendet werden sollen. Wird keine Option angegeben, so wird der Weg für Typ 1 berechnet. Bei Angabe der Option `-h` soll eine Usage-Meldung ausgegeben werden.

Implementieren Sie zumindest drei verschiedene Arten von Verhalten eines Roboters. Sie können das Verhalten frei wählen, sichergestellt werden muss jedoch, dass ein Roboter garantiert einen Ausgang findet, sofern einer existiert. Wie schnell ein Roboter einen Ausgang findet, ist nicht relevant. Es gibt eine Roboter-Basisklasse, von der die verschiedenen Arten von Robotern erben sollen. Es soll eine virtuelle Methode implementiert werden, die jeweils ein bestimmtes Verhalten des Roboters bei der Suche implementiert. Durch die verschiedenen Implementierungen dieser virtuellen Methode soll ein unterschiedliches Verhalten der Roboter erreich werden können. Dies bedarf auch der entsprechenden Verwendung der Roboterobjekte im Programm. Ein Roboter soll die Anzahl der Schritte mitrechnen, die er benötigt, um vom Ein- zum Ausgang zu gelangen. Ein Schritt ist das Bewegen von einem der Felder im Labyrinth zu einem anderen Feld.
In der Implementierung des Verhaltens eines Roboters soll keine Zufallsfunktion verwendet werden. Der vom jeweiligen Robotertyp gefundene Weg muss daher stets derselbe Weg sein. Unterschiede können nur bei verschiedenen Robotertypen auftreten. Wenn Sie Ihr Programm mit zusätzlicher Funktionalität versehen wollen, so machen Sie diese durch selbstgewählte zusätzliche Kommandozeilenoptionen verfügbar. Werden diese zusätzlichen Parameter nicht angegeben, so soll das Programm das normale Verhalten zeigen.

4. Parallele Implementierung mit Threads

Mehrere Roboter sollen gleichzeitig im gleichen Labyrinth nach einem Ausgang suchen können. Robotertypen können gemischt werden. Jeder Roboter wird durch einen Thread implementiert. Die Threads müssen einander nicht erkennen beziehungsweise blockieren, sondern kopieren sich das ursprüngliche Labyrinth in ein eigenes Attribut. Jeder Roboter sucht unabhängig von den anderen Robotern seinen Weg. Falls ein Weg existiert, muss dieser auch gefunden werden. Bei Erreichen des Ziels, soll jeder Prozess die Anzahl der Schritte, die benötigt wurden, ausgeben.

5. Testdateien

Auf Moodle sind diverse Testdateien zu finden, die verschiedene Varianten von Labyrinthen enthalten:

- maze1_small.txt: ein sehr kleines Labyrinth, ideal für erste Tests.
- maze2_unicursal.txt: ein größeres Labyrinth mit genau einem sehr langen Weg.
- maze3_braid.txt: ein größeres Labyrinth mit mehreren Wegen.
- maze4_braid.txt: ein sehr großes Labyrinth mit mehreren Wegen.
- maze5_cavern.txt: ein größeres Labyrinth mit freu verlaufenden Wegen (nur zu Testzwecken für Fortgeschrittene).

Obige Labyrinthe wurden mit dem Programm Daedalus 2.3 erzeugt. Dieses Programm ist frei verfügbar unter: http://www.astrolog.org/labyrnth/daedalus.htm (Windows OS) und kann verwendet werden, um weitere Testdateien zu erzeugen. Das Programm erlaubt auch die verwendeten ASCII-Dateien einzulesen und ein Labyrinth in einer einfachen 3D-Graphik zu begehen.

Beachten Sie, dass die Testdateien Windows-Text-Dateien sind und behandeln Sie den Unterschied in der Darstellung von Newline-Character entsprechend.