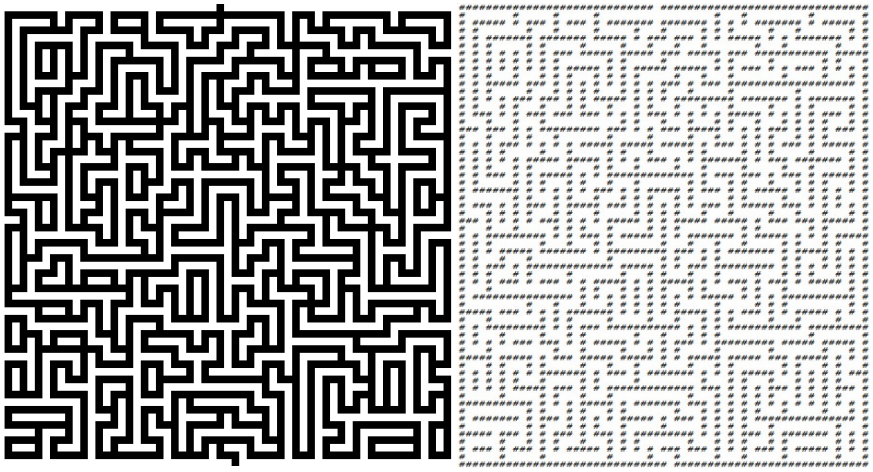
ÜBUNG 6+7

Roboter-Labyrinth

Schreiben Sie ein C++ Programm, das für ein gegebenes Labyrinth einen Weg vom Eingang zum Ausgang berechnet. Design und Implementierung erfolgt schrittweise über drei Übungsabgaben hinweg.

Überblick

Übung 6: Design der Klassen und Datenstrukturen, Einlesen des Labyrinths, Implementierung eines Roboters, der einen Weg im Labyrinth sucht. Das Programm soll das Labyrinth mit dem gefundenen Weg ausgeben können. Mögliche Visualisierungen für die Markierung der Wege werden in der Übung diskutiert.
Übung 7: Mehrere Roboter mit Threads: Es sollen nun mehrere Roboter auf die Suche nach dem Ausgang geschickt werden können.



Oben gezeigt ist (links) die Visualisierung eines Labyrinths und dessen Darstellung im ASCII Format der Datei (rechts), wie sie von Ihrem Programm eingelesen und dargestellt werden können soll (ohne Weg).

Dateiformat: Ein Feld wird durch genau ein ASCII Zeichen dargestellt, dieses stellt entweder eine Mauer dar (‚#') oder ein freies Feld (Leerzeichen). Ein Weg hat stets genau die Breite eines einzelnen Zeichens. 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 andere (zweite) Öffnung ist der Ausgang (Ein- und Ausgang sind nicht speziell markiert). Verzweigungen sind nur horizontal und vertikal möglich, also in höchstens 4 Richtungen.

Übung 6: Objektorientiertes Design und E/A-Operationen

Entwerfen Sie Klassen, die erlauben verschiedene Roboter und das Labyrinth als Objekte zu repräsentieren. Entwerfen Sie das Design der Schnittstellen. Sämtliche Klassen sollen in Header-Dateien spezifiziert werden.
Implementieren sie die Kommandozeile und die Ein/Ausgabeoperationen für das Labyrinth. D.h. Sie sollen ein Labyrinth einlesen können und es am Bildschirm ausgeben können. Kommandozeile:

```
labrob DATEINAME [-t1] [-t2] [-t3]...[-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, -t2, -t3 bis -tN“ sind optional. Ab Abgabe 6 werden diese verwendet, um die Typen der Roboter anzugeben. Wird keine Option angegeben, so wird der Weg für Typ 1 (t1) berechnet. Bei Angabe der Option „-h“ soll eine Usage-Meldung ausgegeben werden.

Abzugeben sind die Header-Dateien mit den Klassendeklarationen und die Implementierung der Kommandozeile und der E/A-Operationen. Die Header-Dateien müssen bei Aufruf von 'make all' eingebunden werden und das Programm ohne Fehler übersetzt werden können. Hinweis: wenn auf Variablen oder Funktionen einer Klasse nicht zugegriffen wird, so kann die Vereinbarung der Klasse übersetzt werden, auch wenn keine Implementierungen der Funktionen vorliegen.

Übung 6: Objektorientierte Implementierung

Implementieren Sie zumindest drei verschiedene Arten von Verhalten eines Roboters (bzw. dessen Suchalgorithmus). 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.
Das Verhalten der Roboter soll objektorientiert implementiert werden, indem eine Basisklasse „Roboter“ definiert wird, und zumindest drei verschiedene Roboter-Klassen von dieser Klasse erben. Dadurch entsteht eine Klassenhierarchie von Roboterklassen. 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 erreicht werden können. Dies bedarf auch der entsprechenden Verwendung der Roboterobjekte im Programm.
Es sollen Objekte der Roboter erstellt werden und in einer polymorphen Liste gespeichert werden. Das Programm arbeitet dann die Liste dieser Roboter ab, und schickt jeden dieser Roboter, einen nach dem anderen, auf die Suche im Labyrinth, bis er sein Ziel gefunden hat. Ein Roboter soll die Anzahl der Schritte mitrechnen, die er benötigt, um von Ein- zum Ausgang zu gelangen. Ein Schritt ist das Bewegen von einem der Felder im Labyrinth zu einem anderen Feld.

Nachdem alle Roboter ein Ziel gefunden haben, soll eine Tabelle ausgegeben werden, in der für jeden Roboter angegeben wird, wie viele Schritte er benötigt hat, um vom Eingang zum Ausgang zu gelangen.
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 (da diese verschiedenes Verhalten haben).

Entscheiden Sie, wie Sie eine gute Darstellung der Roboter verschiedenen Typs erreichen können.
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 Verhalten zeigen, wie hier beschrieben.

Übung 7: Parallele Implementierung mit Threads

Erweitern Sie nun Ihr Programm, sodass mehrere Roboter gleichzeitig im selben Labyrinth nach einem Ausgang suchen können. Robotertypen können gemischt werden. Jeder Roboter wird durch einen Thread implementiert. Die Threads müssen einander nicht erkennen bzw. blockieren können, 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, auf stdout ausgeben.

ALLGEMEINES

Testdateien

Auf Moodle sind diverse Testdateien zu finden, die verschiedene Varianten (Größe und Eigenschaften) von Labyrinthen enthalten:

- „maze1_small.txt“: ein sehr kleines Labyrinth, ideal für erste Tests.
- „maze2_unicursal.txt“: ein größeres Labyrinth mit genau einem (sehr langem) 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 frei verlaufenden Wegen (nur zu Testzwecken für Fortgeschrittene, ein Weg kein breiter als ein Feld sein)

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-Grafik zu begehen.

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

Makefiles

Verwenden Sie für die Übungen Makefiles mit getrennten Compiler und Linker Einträgen.

Abgabe

Abzugeben ist ein .tgz-File mit

1.) Das Makefile

2.) den C++ Headerdateien und Sourcedateien inkl. Kommentaren

3.) README Datei, die eine kurze Beschreibung Ihres Programms enthält. Daraus soll insbesonders hervorgehen, wie Roboter im Labyrinth dargestellt werden.

Das Projekt muss mittels ‚make all' übersetzt werden können.

Abgaben 5+6+7 über Moodle. Den Abgabetermin entnehmen Sie dem jeweiligen Moodle Eintrag „Abgabe 5", „Abgabe 6" und „Abgabe 7".