Hinweise zur Bearbeitung und Abgabe

- Bitte nutzen Sie MARS zum Simulieren Ihrer Lösung. Stellen Sie sicher, dass Ihre Abgabe in MARS ausgeführt werden kann.
- Sie erhalten für jede Aufgabe eine separate Datei, die aus einem Vorgabe- und Lösungsabschnitt besteht. Ergänzen Sie bitte Ihren Namen und Ihre Matrikelnummer an der vorgegebenen Stelle. Bearbeiten Sie zur Lösung der Aufgabe nur den Lösungsteil unterhalb der Markierung:

```
#+ Loesungsabschnitt
#+ ----------------
```

- Ihre Lösung muss auch mit anderen Eingabewerten als den vorgegebenen funktionieren. Um Ihren Code mit anderen Eingaben zu testen, können Sie die Beispieldaten im Lösungsteil verändern.
- Bitte nehmen Sie keine Modifikationen am Vorgabeabschnitt vor und lassen Sie die vorgegebenen Markierungen (Zeilen beginnend mit #+) unverändert.
- Auch wenn die Simulation für Ihre Lösung korrekte Ergebnisse liefert, kann die Lösung Fehler enthalten. Eine korrekte Lösung muss auch die bekannten **Registerkonventionen** einhalten!
- Bitte gestalten Sie Ihren Assemblercode nachvollziehbar und verwenden Sie detaillierte Kommentare, um die Funktionsweise Ihres Assemblercodes darzulegen.
- Wir untersuchen alle Abgaben auf Plagiate. **Plagiate sind Täuschungsversuche und führen zur Bewertung „nicht bestanden“ für die gesamte Modulprüfung. Die aktuell bestehende Freiversuchsregelung an der TU Berlin greift in diesem Fall nicht.**
- Die Abgabe erfolgt über ISIS. Laden Sie die zwei Abgabedateien separat hoch.

Aufgabe 1: Quadratur-Dekodierung (10 Punkte)

Drehgeber sind mechanische Sensoren, die Drehwinkel messen können. Sie werden beispielsweise bei Bedienelementen wie Lautstärke-Drehreglern und in Maschinen eingesetzt. Abbildung 1 zeigt einen einfachen Drehgeber, der die Drehung einer mechanischen Welle in 90°-Schritten erfassen kann. An der Welle ist eine Scheibe angebracht, auf der zwei um 90° versetzte Ringe je zur Hälfte schwarz und weiß markiert sind. Zwei Lichtsensoren messen an einer festen Position die aktuell sichtbare Markierung der beiden Ringe und erzeugen dadurch die zwei Ausgangssignale A und B. Diese werden als einzelne Bits von einem Computer eingelesen, wobei schwarze Abschnitte durch eine 1 und weiße Abschnitte durch eine 0 repräsentiert werden.

Um die Rotationsbewegung der Achse nachzuvollziehen, werden A und B periodisch ausgelesen und in zwei Arrays gespeichert. Sofern häufig genug gemessen wird, lässt sich aus den Werten in den Arrays die Gesamtdrehung zwischen Anfang und Ende der Messreihe errechnen. Eine Beispiel-Messreihe mit Auswertung ist in Tabelle 1 zu sehen.

Implementieren Sie die Funktion `qdec`, welche die Gesamtdrehung aus einer Messreihe von 16 Sensorwert-Paaren berechnet und den Gesamtdrehwinkel in Grad zurückgibt. Die Sensorwerte von A und B sind in den Arrays `array_a` und `array_b` als vorzeichenlose Bytes (`unsigned char` in C) gespeichert. Der Zustand von A und B zum Zeitpunkt 0 ist als nicht gedreht (0°) anzusehen. Ganze Umdrehungen

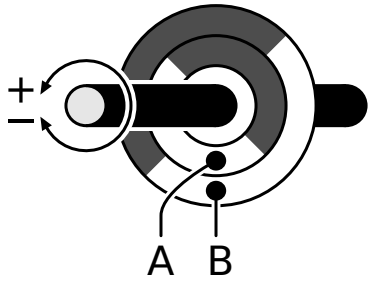


Abbildung 1: Drehgeber mit quadraturkodiertem Ausgangssignalpaar A, B

Zeitpunkt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$A =$	0	0	1	1	0	1	1	1	0	0	1	1	1	1	1	1
$B =$	0	1	1	0	0	0	1	1	1	0	0	1	0	1	1	0
Winkeländerung	–	$+90°$	$+90°$	$+90°$	$+90°$	$-90°$	$-90°$	$+0°$	$-90°$	$-90°$	$-90°$	$-90°$	$+90°$	$-90°$	$+0°$	$+90°$
Drehwinkel	$0°$	$90°$	$180°$	$270°$	$360°$	$270°$	$180°$	$180°$	$90°$	$0°$	$-90°$	$-180°$	$-90°$	$-180°$	$-180°$	$\mathbf{-90°}$

Tabelle 1: Beispiel-Messreihe mit Auswertung. Der Gesamtdrehwinkel beträgt $-90°$.

sollen mitgezählt werden, sodass auch Ergebnisse größer als $360°$ oder kleiner als $-360°$ möglich sind. Drehungen gegen den Uhrzeigersinn zählen negativ, Drehungen im Uhrzeigersinn positiv.

Signatur der zu implementierenden Funktion:

```
int  qdec(  unsigned char *array_a,  unsigned char *array_b);
```

`$v0`	`$a0`	`$a1`

Sie dürfen annehmen, dass die beiden Eingabe-Arrays nur die Werte 0 und 1 enthalten. Die Länge der beiden Arrays ist auf 16 festgelegt. Falls A und B zeitgleich den Wert ändern, ist ein Messfehler aufgetreten. In diesem Fall soll `qdec` den Wert -1 zurückgeben, da eine Winkelbestimmung nicht mehr möglich ist.

Tipps:

- Stellen Sie zur Veranschaulichung eine Tabelle mit allen möglichen Kombinationen von aktuellen und vorherigen Werten von A und B auf. Notieren Sie, ob es sich um gültige Zustandsübergänge handelt und um welchen Wert sich der Drehwinkel gegebenenfalls geändert hat. Welche logischen Operationen könnten zur Ermittlung der Winkeländerung genutzt werden?

Aufgabe 2: Wegsuche im Irrgarten (10 Punkte)

Eine häufig zu findende Problemstellung in der Informatik ist die Suche nach einem möglichst kurzen Weg in einer zweidimensionalen Struktur. Beispiele sind Bewegungen von Spielfiguren in Computerspielen oder das automatische Verdrahten von elektronischen Komponenten auf Leiterplatten oder Chips. Der *Lee-Algorithmus* ist ein Lösungsansatz zu dieser Problemstellung. Um einen Weg zwischen den Feldern S und D im Irrgarten zu finden, wird in zwei Schritten vorgegangen: Wellenausbreitung und Rückverfolgung.

Die Aufgabe dieser Hausaufgabe ist es, in einem Irrgarten den ersten Schritt der **Wellenausbreitung**

zu implementieren. Dabei sollen alle vom Startfeld S erreichbaren freien Felder mit der Entfernung zum Startfeld markiert werden. Der zweite Schritt (Rückverfolgung) ist nicht Teil dieser Hausaufgabe.

Um Irrgärten darzustellen, verwenden wir ein Array von $8 \times 8 = 64$ Feldern. Abbildung 2a zeigt, wie Array-Indizes zu Feldern im Irrgarten korrespondieren. Die Elemente des Arrays sind vorzeichenlose Bytes. Zu Beginn der Wellenausbreitung weisen leere Felder im Irrgarten den Wert 0 auf, Hindernis-Felder den Wert 255, siehe Tabelle 2.

Implementieren Sie die Funktion `prop_wave`, welche im Irrgarten `maze` vom Feld `pos` ausgehend rekursiv die freien Felder mit Entfernungswerten markiert. Die Entfernungswerte sollen die bestehenden Werte im Array ersetzen. Hindernisse dürfen nicht mit Entfernungswerten markiert werden. Da der Wert 0 unmarkierte Felder kodiert, soll das Startfeld den Wert 1 zugewiesen bekommen. Der Entfernungswert von direkten Nachbarn des Startfelds soll 2 betragen. Werden Felder mehrfach besucht, soll die kleinstmögliche Entfernung gespeichert werden, sodass mit dem Ergebnis der Wellenausbreitung die kürzesten Wege zum Startfeld S gefunden werden können.

Die Vorgabedabei ruft `prop_wave` mit `dist`=1 und dem Startfeld S im Argument `pos` auf und gibt danach den Irrgarten als Text aus. Eine korrekte Lösung liefert für den vorgegebenen Irrgarten `maze1` das in Abbildung 2d dargestellte Ergebnis.

Für alle Nachbarfelder des aktuellen Felds soll sich `prop_wave` **rekursiv** selbst aufrufen. Für die rekursiven Aufrufe muss `dist` um 1 erhöht werden. Die Rekursion ist abzubrechen, wenn es sich bei dem aktuellen Feld um ein Hindernis handelt oder wenn das aktuelle Feld bereits mit einem Entfernungswert kleiner oder gleich `dist` markiert ist.

Verwenden Sie die im Vorgabecode implementierte **Hilfsfunktion `neighbor`**, um ausgehend von einem Feld `pos` die Indizes der Nachbarfelder zu berechnen. Der Paramater `direction` gibt dabei die Richtung des gesuchten Nachbarfelds an: 0=oben, 1=links, 2=unten, 3=rechts. `neighbor` gibt den Index des entsprechenden Nachbarfelds zurück, falls ein solches Nachbarfeld existiert. Falls das Nachbarfeld nicht existiert, gibt `neighbor` -1 zurück. Signatur der Hilfsfunktion `neighbor`:

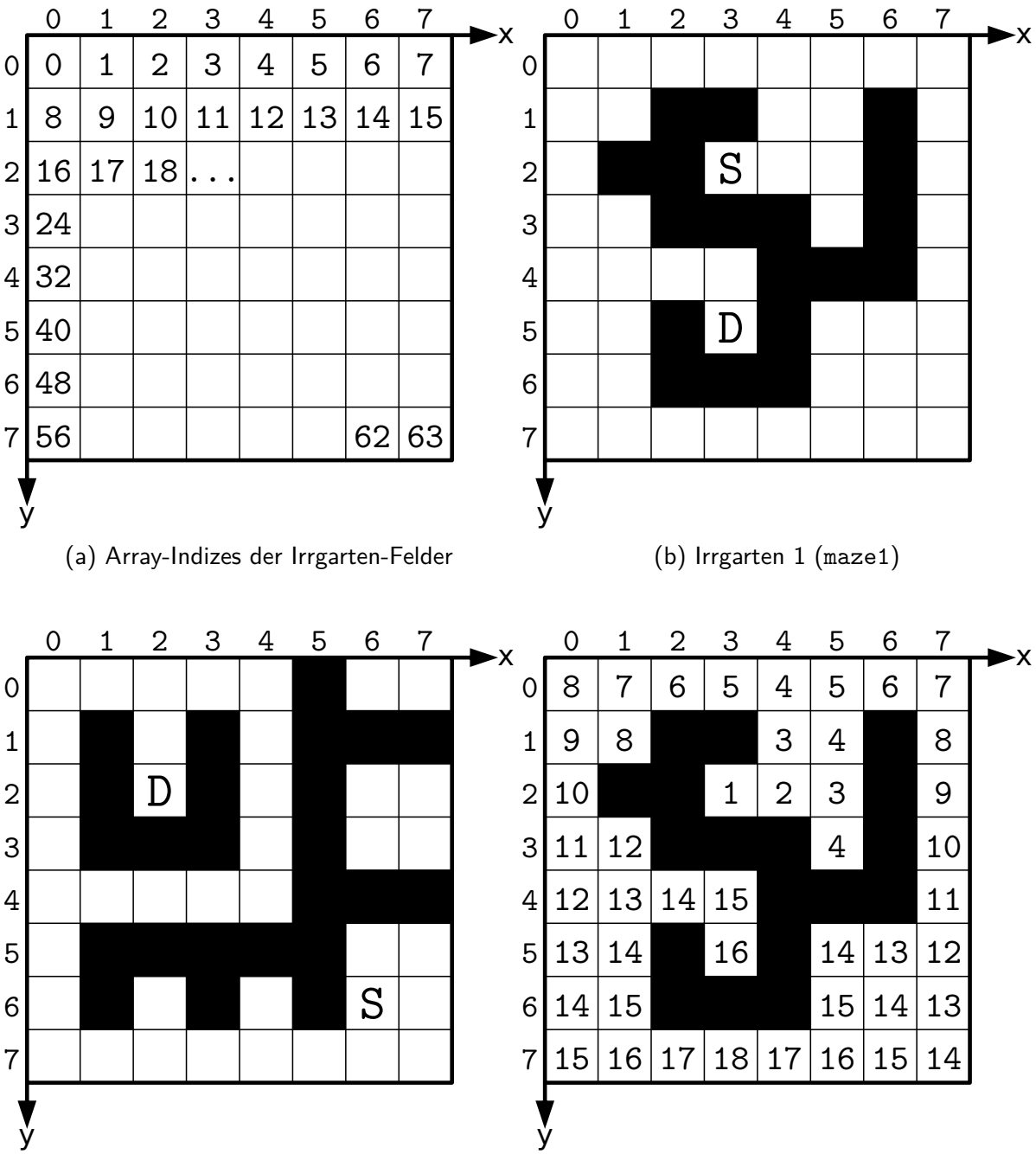
`int neighbor( int pos, int direction);`		
`$v0`	`$a0`	`$a1`

Signatur der zu implementierenden Funktion:

`void prop_wave( unsigned char *maze, int pos, int dist);`			
`$v0`	`$a0`	`$a1`	`$a2`

Tipps:

- Nutzen Sie den Befehl `lbu`, um Werte aus dem Array in Prozessorregister zu laden.
- Sie können sich die Fehlersuche erleichtern, indem Sie vorübergehend die Rekursionstiefe oder die Suchrichtung einschränken (z. B. Maximalentfernung 5 oder nur nach rechts).
- Berücksichtigen Sie, dass Randfelder nur 2 oder 3 Nachbarn haben.



(a) Array-Indizes der Irrgarten-Felder

(b) Irrgarten 1 (`maze1`)

(c) Irrgarten 2 (`maze2`)

(d) Erwartetes Ergebnis für Irrgarten 1 (`maze1`)

Abbildung 2: Beispiel-Irrgärten sowie Veranschaulichung der Array-Offsets. Die leeren Felder sind frei, die ausgefüllten Felder repräsentieren Hindernisse. *S* und *D* markieren die Start- und Zielfelder.

Wert	**Hindernis**	**Abstands-Markierung**	**Darstellung**
0	nein	nein	leer
1–99	nein	ja, Wert = Abstand	als Zahl
255	ja	nein	`XX`

Tabelle 2: Kodierung der Felder im Irrgarten durch vorzeichenlose Bytes