Technische Hochschule Ingolstadt
Fakultät für Informatik

Praktikum zu „Grundlagen der Programmierung“

Blatt 3

Lernziele: Steuerung der Bewegung eines Zeichens auf dem Bildschirm
Erforderliche Kenntnisse: Übergabe von Parametern an Funktionen (*by-value*, *by-pointer*)
Voraussetzungen:

- Wir nehmen im Folgenden an, dass Sie die virtuelle Linux-Maschine benutzen, die im Rahmen der Vorlesung zur Verfügung gestellt wird.

Übersicht

Im Rahmen dieses Aufgabenblatts wird erstmals ein Wurm im Fenster der Konsole bewegt. Die Richtung des Wurms kann durch die Pfeiltasten der Tastatur beeinflusst werden. Eine genauere Beschreibung des Leistungsumfangs und der Steuerungsmöglichkeiten durch Tasten finden Sie im Verzeichnis `Worm005` in den Dateien `Readme.fabr` und `usage.txt`.
Abb.~1 zeigt die Ausführung des Programms `bin/worm`

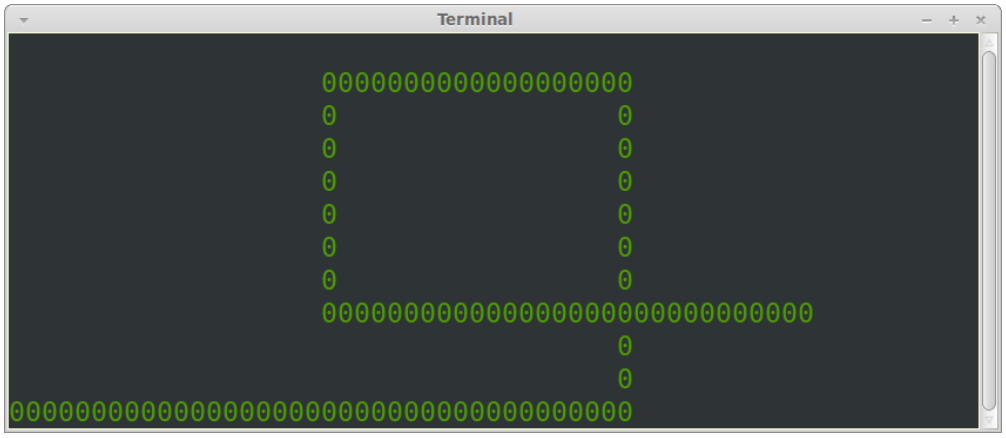

Abbildung 1: Ausführung des Programms `worm` auf Linux

Wie schon im letzten Aufgabenblatt wird es auch diesmal Ihre Aufgabe sein, ein als Gerüst vorgegebenes Programm zu vervollständigen. Im Fokus dieses Aufgabenblatts befinden sich folgende Themen:

- Verwendung von Präprozessor-Konstanten (#define).
- Verwendung globaler Variablen.
- Strukturierung eines Programms durch Funktionen.
- Verwendung von Funktionsparametern (Wert-Parameter, Resultat-Parameter).

- Bewegung/Steuerung eines Zeichens im Ausgabefenster durch periodisches Abfragen der Tastatur und Aktualisierung des virtuellen Ausgabepuffers für das Fenster der Konsole.

Aufgrund Ihrer fehlenden Erfahrung in der C-Programmierung sind die von Ihnen einzubringenden Fragmente noch sehr klein. Versuchen Sie aber unbedingt, alle Teile des Programms zu verstehen!

Vorbereitung: Virtuelle Maschine starten, anmelden und Repository von Bitbucket klonen.

Hinweis: Auf allen nachfolgenden Blättern wird dieser Hinweis zur Vorbereitung nicht mehr explizit erwähnt!

Aufgabe 1 (Kopieren, Auspacken und Umbenennen des Templates)

Im Moodle-Kursraum finden Sie im Verzeichnis Praktikum die Datei `Worm005Template.zip`.
Laden Sie diese Datei im Browser herunter nach `~/Downloads`.
Öffnen Sie sodann eine Shell.
In der Shell wechseln Sie in das Verzeichnis `~/GdP1/Praktikum/Code`.
Falls es dieses Verzeichnis bei Ihnen nicht gibt, legen Sie es bitte zuvor mittels `mkdir` an.

```
$ cd ~/GdP1/Praktikum/Code
$ cp ~/Downloads/Worm005Template.zip .
```

Vergewissern Sie sich mittels des Befehls ls, dass die Zip-Datei im Verzeichnis vorhanden ist.

```
$ ls Worm005Template.zip
```

Entpacken Sie die Zip-Datei durch den nachfolgenden Befehl. Dadurch wird das Verzeichnis `Worm005Template` mit einigen Dateien darin angelegt.

```
$ unzip Worm005Template.zip
```

Listen Sie den Inhalt des neuen Unterverzeichnisses.

```
$ ls -la Worm005Template
```

Benennen Sie das Verzeichnis `Worm005Template` um in `Worm005`.

```
$ mv Worm005Template Worm005
```

Listen Sie den Inhalt des umbenannten Unterverzeichnisses.

```
$ ls -la Worm005
```

Listen Sie den Inhalt des aktuellen Verzeichnisses.

```
$ ls -la
```

Löschen Sie das Zip-Archiv.

```
$ rm -f Worm005Template.zip
```

Aufgabe 2 (Vervollständigung der Datei worm.c)

Öffnen Sie **zwei** Shells und wechseln Sie in jeder dieser Shells ins Verzeichnis
`~/GdP1/Praktikum/Code/Worm005`

Hinweis: Die Tabulator-Taste sorgt für die praktische Vervollständigung von Pfad- und Dateinamen.

Nun kommen wir zur eigentlichen Aufgabe dieses Blattes. In der vorherigen Aufgabe haben Sie unter anderem die Datei `worm.c` kopiert. Diese Datei enthält schon das wesentliche Gerüst eines C-Programms, das die eingangs erwähnte Aufgabenstellung für dieses Übungsblatt löst. Es fehlen lediglich noch einige Anweisungen im Programm, die Sie hinzufügen sollen.

Öffnen Sie nun die Datei `worm.c` in einem Editor, damit wir unsere Tour durch die Datei beginnen können.

Abschnitt Header-Dateien

Zeilen 11-16: Es werden neben der Header-Datei `curses.h` noch weitere Header-Dateien der C-Bibliothek eingebunden. Dadurch werden dem Compiler Deklarationen von Konstanten, Makros und Funktionen dieser Bibliotheken bekannt gemacht.
Informationen zu den einzelnen Bibliotheken finden Sie online unter:

- http://invisible-island.net/ncurses/man/ncurses.3x.html
- http://www2.hs-fulda.de/~klingebiel/c-stdlib
- http://manpages.ubuntu.com/manpages/bionic/en/man7/unistd.h.7posix.html

Abschnitt Präprozessor-Konstanten

Zeilen 18 - 55: (Überblick)
In diesem Abschnitt des Programms werden diverse Präprozessor-Konstanten mittels `#define` definiert. Durch die Einführung von Konstanten wird die Lesbarkeit und Wartbarkeit von Programmen erhöht. Statt expliziter Zahlen oder Zeichenketten können wir symbolische Namen als Platzhalter im Programmtext verwenden. Aussagekräftige Namen für Konstanten erleichtern das Verständnis des Programms. Soll eine im Programmtext mehrfach vorkommende Zahl geändert werden, reicht es die Deklaration der Präprozessor-konstanten zu ändern.
Die Aufgabe und Funktionsweise des Präprozessors wird in der Vorlesung besprochen.

Zeile 35: (Konstante `COLP_USER_WORM`)
Im vorliegenden Programm möchten wir den Wurm in grüner Farbe vor schwarzem Hintergrund darstellen. Damit bei der Ausgabe eines Wurmsegments nicht jedesmal diese beiden Farben erwähnt werden müssen, erlaubt die Curses-Bibliothek die Verwendung von Farbpaaren (*color pairs*). Farbpaare müssen vor ihrer Verwendung definiert werden[1] und durch eine eindeutige Nummer gekennzeichnet werden. Für die Nummerierung führen wir hier die Nummer 1 ein und geben dem ersten und vorerst einzigen Farbpaar den Namen `COLP_USER_WORM`.

[1] Wir tun dies in der Funktion `initializeColors`.

Zeile 38: (Konstante `SYMBOL_WORM_INNER_ELEMENT`)
Derzeit werden alle Segment des Wurms mit dem gleichen Symbol ‚`O`‘dargestellt. Später werden wir spezielle Symbole für den Kopf und das letzte Element des Wurms benutzen.

Zeilen 41-43: (Konstanten für den Spielzustand)
Während des Spiels befinden wir uns normalerweise im Zustand `WORM_GAME_ONGOING`. Falls der Benutzer aber durch Unachtsamkeit den Wurm über den Rand des Fensters steuert, wechselt das Spiel in den Zustand `WORM_OUT_OF_BOUNDS`. Falls der Benutzer während des Spiels die Taste ‚**q**‘drückt, wechselt das Spiel in den Zustand `WORM_GAME_QUIT`.

Zeilen 46-49: (Konstanten für die Steuerung der Bewegungsrichtung)
Der Benutzer kann durch Betätigung der Pfeiltasten auf der Tastatur die Bewegungsrichtung des Wurms beeinflussen. Um die gewünschte Richtung zwischen den einzelnen Funktionen zu signalisieren, führen wir hier vier Konstanten für die möglichen Bewegungsrichtungen ein.

Hinweis: Im Rahmen der Bearbeitung des nächsten Aufgabenblatts (Blatt 4) werden wir die meisten der gerade eingeführten Konstanten durch spezielle Aufzählungstypen (`enum`) ersetzen, was die Lesbarkeit des Programms erhöhen und seine Fehleranfälligkeit deutlich senken wird.

Abschnitt Globale Variablen

Um den Wurm gemäß den Richtungsvorgaben durch den Benutzer über den Bildschirm bewegen zu können, müssen wir im Programm mehrere Eigenschaften des Wurms speichern und laufend verändern. Zu diesen Eigenschaften (Attributen) des Wurms gehören mindestens seine aktuelle Position und seine aktuelle Bewegungsrichtung. Diese Eigenschaften können auf viele Arten in der Sprache C modelliert werden.
Weil es uns noch an wesentlichen Kenntnissen über zusammengesetzte Datenstrukturen mangelt, fangen wir mit einer sehr einfachen Modellierung durch mehrere globale Variablen an. Später werden wir jedoch für den Wurm eine eigene Datenstruktur einführen, um unsere Modellierung besser zu kapseln und flexibler zu machen.
Der Einsatz von globalen Variablen macht den Zugriff auf die Eigenschaften des Wurms sehr einfach, da globale Variablen automatisch in allen Funktionen des Programms sichtbar sind. Insbesondere können die Eigenschaften in allen Funktionen direkt per Zuweisung an die globalen Variablen manipuliert werden.
Leider führt aber gerade die Verwendung von globalen Variablen zu unflexiblen und schwer zu wartenden Programmen!

Zeilen 56-57: (Aktuelle Position)
Wir speichern die Position des Kopfelements (Definiert über Zeile und Spalte) in den beiden Variablen `theworm_headpos_y` und `theworm_headpos_x`.

Zeilen 61-62: (Aktuelle Richtung als Delta/Offset für Koordinaten)
Wir speichern die aktuelle Richtung des Wurms als „Delta der Position pro Schritt“in den beiden Variablen `theworm_dx` und `theworm_dy`.
Es handelt sich hier um eine spezifische Implementierungsentscheidung.

Wenn der Wurm sich gerade an Position (y,x) befindet und zum Beispiel nach rechts kriechen soll, so müssen wir irgendwann die im nächsten Bewegungsschritt folgende neue Kopfposition berechnen, indem wir zur aktuellen x-Koordinate (Spalte) den Wert +1 addieren und zur aktuellen y-Koordinate (Zeile) den Wert +0 addieren. Wir addieren also in der Horizontalen (x) das Delta +1 und in der Vertikalen (y) das Delta +0.
Dieser Gedankengang erklärt die Benennung der Variablen:

- `theworm_dx` für das Delta in x-Richtung
- `theworm_dy` für das Delta in y-Richtung

Für die vier möglichen Bewegungsrichtungen ergeben sich folgende Deltas (Offsets):

- Nach rechts: $\texttt{theworm\_dx} = +1, \texttt{theworm\_dy} = 0$
- Nach links: $\texttt{theworm\_dx} = -1, \texttt{theworm\_dy} = 0$
- Nach oben: $\texttt{theworm\_dx} = 0, \texttt{theworm\_dy} = -1$
- Nach unten: $\texttt{theworm\_dx} = 0, \texttt{theworm\_dy} = +1$

Hinweis: Die Übersetzung der Bewegungsrichtungen `WORM_UP`, `WORM_DOWN`, ... in die hier angesprochenen Offsets für Koordinaten erfolgt in der Funktion `setWormHeading()`.

Zeile 64: (Farbe)
Im Vorgriff auf zukünftige Verschönerungen der Ausgabe modellieren wir auch die ‚aktuelle' Farbe des Wurms in der globalen Variable `theworm_wcolor`. Es handelt sich hierbei um die Nummer eines Farbpaars bestehend auf Vordergrund- und Hintergrundfarbe.

Abschnitt Vorwärtsdeklaration von Funktionen

In C müssen Funktionen vor ihrer Verwendung deklariert werden. Um dieses Gebot einzuhalten, kann man entweder mühsam versuchen, alle Funktionsdefinitionen in der richtigen Reihenfolge anzuordnen, oder aber man nutzt die Möglichkeit der Vorwärtsdeklaration. Ein Vorwärtsdeklaration führt einfach den Kopf der Funktion (die Signatur der Funktion) bestehend aus Resultattyp, Name und Parameterliste auf.
Die eigentliche Definition der Funktion, bestehend aus Signatur und Funktionsrumpf, kann dann an beliebiger Stelle im Rest des Programms ausgeführt werden. Wir nutzen diese Technik in den Zeilen 66 – 92.

Abschnitt Funktionsdefinitionen

Nun müssen Sie aktiv werden. Dieser Abschnitt enthält die Definitionen aller Funktionen des Programms. Die Funktion `main()`, mit der die Ausführung eines jeden C-Programms beginnt, wird zuletzt definiert.
In nahezu allen Funktionen fehlen kleine Fragmente, die durch einen Platzhalter `@nnn` gekennzeichnet sind.
Im Folgenden werden wir gezielt die von Ihnen zu ergänzenden Fragmente ansprechen, indem wir auf den jeweiligen Platzhalter `@nnn` Bezug nehmen.

Beginnen wir die Vervollständigung mit der Funktion `main()`, die ähnlich wie auf Blatt 2 aufgebaut ist.

Funktion `main`:
Nachdem wir die Curses-Anwendung mittels `initializeCursesApplication` initialisiert haben (Zeile 356), definieren wir in `initializeColors` die zu verwendenden Farbpaare (Zeile 357). In Zeile 364 wird geprüft, ob unser Konsolenfenster eine gewisse Mindestgröße hat. Falls das nicht der Fall ist, geben wir eine Fehlermeldung aus (Zeilen 368 – 371) und speichern `RES_FAILED` als Resultat-Code in der Variable `res_code`.
Falls aber die Fenstergröße ausreicht, rufen wir die Funktion `doLevel` auf (Zeile 373), welche die eigentliche Hauptschleife des Programms enthält. Die Funktion `doLevel` liefert einen Resultat-Code, den wir ebenfalls abspeichern, um dann schlussendlich das Terminal wieder zurück in einen normalen Zustand zu versetzen (Zeile 374).

`@001`: Als letzte Anweisung der Funktion `main` geben wir den Resultat-Code zurück. (Welchen?)

Funktion `initializeColors`:
In dieser Funktion werden die im Programm verwendeten Farbpaare (Vordergrund/Hintergrund) definiert. Als erstes muss die Funktion `start_color()` der Curses-Bibliothek aufgerufen werden, um die Verwendung von Farbpaaren zu ermöglichen. Danach können die einzelnen Farbpaarungen mittels der Curses-Funktion `init_pair` eingeführt und über eine eindeutige Nummer zugänglich gemacht werden.
Wir führen nur ein Farbpaar für die Darstellung der Wurmelemente ein und geben ihm die Nummer `COLP_USER_WORM`. Diese Nummer haben wir als Präprozessor-Konstante definiert. Die Curses-Bibliothek stellt bereits Konstanten für die Grundfarben bereit. Die Konstanten heißen: `COLOR_BLACK`, `COLOR_WHITE`, `COLOR_RED`, ...

`@002`: Der Vordergrund eines Wurmelements soll grün sein.

Konsultieren Sie die Dokumentation der Curses-Bibliothek, um die Namen aller zur Verfügung stehenden Farben zu erfahren.

http://invisible-island.net/ncurses/man/curs_color.3x.html

Funktion `getLastRow`:
Diese kleine Hilfsfunktion soll den Index für die letzte benutzbare Zeile auf dem Spielfeld als Resultat zurückliefern. Derzeit besteht unser Spielfeld aus allen Zeilen und Spalten des Konsolenfensters, später werden wir aber einen Teil des Konsolenfensters als Bereich für Statusmeldungen reservieren. Um die genaue Aufteilung des Konsolenfensters in Spielbereich und Statusbereich zu kapseln, führen wir unter anderem die Funktion `getlastRow` ein.
Die Funktion hat keine Parameter und liefert einen Wert vom Typ `int` an den Aufrufer zurück. Das Makro `LINES` der Curses-Bibliothek liefert die Anzahl der zur Verfügung stehenden Zeilen des Konsolenfensters, die erste Zeile (ganz oben) hat den Index 0.

`@003`: Geben Sie den Index der letzten Zeile als Resultat der Funktion zurück.

Funktion `getLastCol`:
Diese kleine Hilfsfunktion soll, analog zur Funktion `getLastRow`, den Index für die letzte benutzbare Spalte auf dem Spielfeld als Resultat zurückliefern.
Die Funktion hat keine Parameter und liefert einen Wert vom Typ `int` an den Aufrufer zurück. Das Makro `COLS` der Curses-Bibliothek liefert die Anzahl der zur Verfügung stehenden Spalten des Konsolenfensters, die erste Spalte (ganz links) hat den Index 0.

`@004`: Geben Sie den Index der letzten Spalte als Resultat der Funktion zurück.

Funktion `setWormHeading`:
Diese Funktion übersetzt eine Richtungsangabe in Form von Präprozessor-Konstanten `WORM_UP`, `WORM_DOWN`, `WORM_LEFT` und `WORM_RIGHT` in Koordinaten-Offsets und speichert diese in den globalen Variablen `theworm_dx` und `theworm_dy`.
Die Bedeutung und Kodierung von Koordinaten-Offsets haben wir bereits oben im Abschnitt über globale Variablen besprochen.
Die Funktion hat einen Wert-Parameter `dir` (*direction*) vom Typ `int` abhängig vom Wert des Parameters werden die der Richtung entsprechenden Offsets für die `x`- und `y`-Koordinate gespeichert. Wir verwenden die Kontrollstruktur `switch` für eine mehrfache Verzweigung. Beachten Sie besonders den Einsatz der Anweisung `break`.

Frage:
Was passiert, wenn keine der Konstanten `WORM_UP`, `WORM_DOWN`, `WORM_LEFT` und `WORM_RIGHT` sondern eine beliebige andere Zahl vom Typ `int` übergeben wird?

Hinweis:
Im Rahmen des nächsten Übungsblatts werden wir die Funktion `setWormHeading` durch die Einführung eines Aufzählungstyps (`enum`) für die Richtungsangaben robuster machen (Thema Typsicherheit).

`@005`: ergänzen Sie die fehlenden Fälle im Code und belegen Sie die globalen Variablen `theworm_dx` und `theworm_dy` mit den richtigen Werten.

Funktion `placeItem`:
Diese Funktion kapselt die Ausgabe eines Zeichens auf dem Spielfeld. Als Wert-Parameter werden die (`y`,`x`)- Koordinaten, das Zeichen (`symbol`) und die gewünschte Farbkodierung (`color_pair`) übergeben.
Der Typ `chtype` des Parameters `symbol` bedarf einer Erklärung. Hierbei handelt es sich um einen Typ-Alias (`typedef`) der Curses-Bibliothek, welcher in der Datei `curses.h` als ein Integer-Typ definiert wird. Der genaue Typ hängt von der Plattform und vom Prozessor-Typ ab. Neben dem Zeichen an sich können über diesen Typ auch zusätzliche Attribute des Zeichens kodiert werden (normal, blinkend, fett, unterstrichen, etc.).
Details erfahren Sie auf der Seite http://invisible-island.net/ncurses/man/curs_attr.3x.html. Nachdem die Schreibposition in Zeile 249 auf die (`y`,`x`)-Koordinaten eingestellt wird, werden mittels `attron` spezielle Attribute der Zeichendarstellung gesetzt. Wir setzen hier die Farben für Vorder- und Hintergrund des Zeichens, indem wir das im Parameter `color_pair` übergebene Farbpaar nutzen. Nach der Ausgabe durch `addch` schalten wir die Attribute mittels `attroff` wieder aus.

Details zu den Funktionen attron/attroff erfahren sie ebenfalls auf der Web-Seite http://invisible-island.net/ncurses/man/curs_attr.3x.html.

`@006`: setzen Sie hier das richtige Argument ein.

Funktion `showWorm`:
Diese Funktion hat die Aufgabe, den Wurm auf dem Bildschirm darzustellen. In der vorliegenden Version des Spiels sehen alle Wurmelemente gleich aus (Konstante `SYMBOL_WORM_INNER_ELEMENT`), und der Wurm hat kein sichtbares Ende, d.h. es werden keine Wurmelemente gelöscht, sondern nur immer neue an der aktuellen Kopfposition gezeichnet.
Aus diesem Grund reduziert sich die Aufgabe der Wurmdarstellung darauf, an der aktuellen Kopfposition ein neues Wurmelement auszugeben. Diese Teilaufgabe erledigt die gerade im letzten Absatz behandelte Funktion `placeItem`.

`@007`: fügen Sie das fehlende Argument im Aufruf von `placeItem` hinzu.

Funktion `initializeWorm`:
Zu Beginn des Spiels müssen die Eigenschaften des Wurms (Position, Richtung und Farbe) initialisiert werden. Diese Aufgabe übernimmt die Funktion `initializeWorm`. Als Parameter erhält die Funktion die Wert-Parameter für die Position des Kopfs (`headpos_y`, `headpos_x`), die Bewegungsrichtung (`dir`) in Form einer der Präprozessor-Konstanten `WORM_UP`, `WORM_DOWN`, `WORM_LEFT`, `WORM_RIGHT` und die Farbe des Wurms (`color`) als Nummer eines Farbpaars.

`@008`: ergänzen Sie die Zuweisung.
`@009`: fügen Sie hier einen geeigneten Aufruf der Funktion `setWormHeading` ein.

Funktion `moveWorm`:
Diese Funktion berechnet aus der aktuellen Position des Kopfelements und der aktuellen Richtung des Wurms die Position des Kopfelements im nächsten Schritt. Alle dazu notwendigen Informationen liegen in dieser Version des Programms als globale Variablen vor.
Allerdings kann es vorkommen, dass der Benutzer durch Unachtsamkeit den Wurm über den Rand des Spielfeldes hinaus steuert, was nicht erlaubt ist und zu einem Abbruch des Programms führen soll.
Allerdings kann es vorkommen, dass der Benutzer durch Unachtsamkeit den Wurm über den Rand des Spielfeldes hinaus steuert, was nicht erlaubt ist und zu einem Abbruch des Programms führen soll.
In der vorliegenden Implementierung wollen wir eine derartige Verletzung der Spielregeln in Form eines Resultat-Parameters an den Aufrufer signalisieren. Der Aufrufer der Funktion übergibt als Argument die Speicheradresse einer Variablen (zu erkennen am Typ `int*` des Parameters `agame_state`), in die die Funktion das Resultat speichern soll. Weil die Adresse einer Variablen übergeben wird und nicht nur ihr Wert, kann die Funktion den Wert der Variablen ändern.
Eine derartige Übergabe von Argumenten an eine Funktion nennt man *Übergabe per Referenz (by-reference) oder Übergabe per Zeiger (by-pointer)*. Die in der Funktion dafür verwendeten formalen Parameter nennt man Resultat-Parameter oder Referenz-Parameter. Syntaktisch wird dies durch den `*` hinter dem Typ (hier `int`) ausgezeichnet.

Ein Referenz-Parameter enthält nicht den Wert einer Variablen sondern deren Adresse. Der Parameter zeigt gewissermaßen auf eine Variable und wird daher *Zeiger (pointer)* genannt.
Das Aufsuchen der Adresse, um den Wert der Variablen zu lesen oder zu schreiben, bezeichnet man als *Dereferenzierung des Zeigers*. Syntaktisch wird dies in C durch Voranstellen eines `*` vor den Namen des Zeigers ausgedrückt, im vorliegenden Fall durch `*agame_state` (z.B. in Zeile 309).
Der Ausdruck `*agame_state` kann gelesen werden als:
„suche die Speicherstelle auf, deren Adresse in `agame_state` steht“

`@010`: ergänzen Sie den Code analog zur vorangehenden Zeile.
`@011`: ergänzen Sie die Fallunterscheidung. Alle Fälle kennzeichnen Fehlersituationen, die analog zur Zeile 309 durch geeignete Belegung des Resultat-Parameters an den Aufrufer signalisiert werden.

Funktion readUserInput:
Die Funktion `readUserInput` kapselt die Abfrage der Tastatureingabe durch den Benutzer.
Die Funktion wird mehrmals pro Sekunde aufgerufen (siehe Funktion `doLevel` unten).
Der Benutzer kann durch Drücken der Pfeiltasten die Bewegungsrichtung des Wurms vorgeben. Darüber hinaus soll das Drücken der Taste ‚s‘den Einzelschrittmodus (*single-step*) einschalten, in dem der Wurm sich nur bewegt, wenn eine Taste gedrückt wird. Dieser Modus (*cheat-mode*) dient zu Testzwecken. Der Einzelschrittmodus soll durch das Drücken der Leertaste ‚~‘wieder beendet werden.
Das Drücken der Taste ‚q‘(*quit*) soll das Programm umgehend beenden. Der Abbruchwunsch des Benutzers wird in der vorliegenden Implementierung durch den Resultat-Parameter `agame_state` an den Aufrufer signalisiert. Wir wählen hier einen Resultat-Parameter, weil wir in späteren Versionen des Spiels noch weitere Resultate aus der Funktion `readUserInput` an den Aufrufer signalisieren werden.
Ob eine Taste gedrückt wird, kann durch Aufruf der Funktion `getch` der Curses-Bibliothek festgestellt werden. Aufgrund der von uns getätigten Einstellungen zur Initialisierung der *curses*-Bibliothek in der Funktion `initializeCursesApplication` blockiert der Aufruf der Funktion `getch` nicht, bis die Eingabe-Taste (Enter/Return/NewLine) gedrückt wird. Details zur Funktion `getch` finden sie auf der Seite

http://invisible-island.net/ncurses/man/curs_getch.3x.html

Falls eine Taste gedrückt wurde, liefert `getch` den Tastencode als positive Integer-Zahl. In der dem Aufruf von `getch` folgenden Mehrfachverzweigung (`switch`) kümmern wir uns um die verschiedenen Fälle.

`@012`: vervollständigen Sie den Code analog zum Fall `KEY_UP`.
`@013`: studieren Sie nochmals die Funktion `initializeCursesApplication`. Dort finden Sie Hinweise, wie Sie in den beiden verbleibenden Fällen das Verhalten von `getch` den Anforderungen entsprechend einstellen können.

Funktion doLevel:
Nun kommen wir zur zentralen und letzten noch verbleibenden Funktion `doLevel`, welche die

komplexeste Funktion in dieser Programmversion ist. Die Funktion `doLevel` enthält die zentrale Steuerung des Programms in Form einer Schleife.
Zu Beginn werden einige Initialisierungen vorgenommen.

Zeile 150: der Spielzustand wird initialisiert auf den Normalzustand `WORM_GAME_ONGOING`.
Zeilen 154-155: die Startposition des Wurms ist links unten.
Zeile 157: die Eigenschaften Position, Bewegungsrichtung und Farbe des Wurms werden initialisiert.
Zeile 163: der Wurm wird an seiner Anfangsposition gezeichnet. Genauer gesagt wird er zunächst in einen von Curses verwalteten virtuellen Fenster-Puffer gezeichnet.
Zeile 166: durch den Aufruf von `refresh` wird der Fenster-Puffer im echten Konsolenfenster angezeigt.
Zeile 169: wir initialisieren die Variable `end_level_loop` mit dem Wert `FALSE`. Die nachfolgende Schleife soll solange ausgeführt werden, wie die Variable `end_level_loop` den Wert `FALSE` hat. Sobald sie den Wert `TRUE` annimmt, soll die Schleife verlassen werden.
Zeile 172: zu Beginn eines jeden Schleifendurchlaufs rufen wir die Funktion `readUserInput` auf, um eventuelle Benutzereingaben abzuholen. Als Argument übergeben wir die Adresse der Variablen `game_state` (siehe Referenz-Parameter der Funktion `readUserInput`).
Falls aufgrund einer Benutzereingabe die Variable `game_state` den Wert `WORM_GAME_QUIT` angenommen hat, soll die Schleife sofort verlassen werden.

`@014`: Belegen Sie die Variable `end_level_loop` mit dem richtigen Wert.

Die nachfolgende Anweisung `continue` sorgt dafür, dass der Rest des aktuellen Schleifendurchlaufs übersprungen wird und sofort die Schleifenbedingung erneut geprüft wird.
Falls `game_state` nicht den Wert `WORM_GAME_QUIT` angenommen hat, ist es nun Zeit, den Wurm einen Schritt zu bewegen.

`@015`: ergänzen Sie das fehlende Argument im Funktionsaufruf. Beachten Sie hierzu die nachfolgende Zeile 182 oder die Zeile 172.
`@016`: reagieren Sie geeignet, falls die Variable durch den Aufruf der Funktion `moveWorm` einen Wert ungleich `WORM_GAME_ONGOING` angenommen hat.

Zeile 187: falls es keinen Grund zum Abbruch gab, kann jetzt der Wurm an seiner neuen Position gezeichnet werden.
Zeile 192: damit das Programm nicht zu schnell läuft, und der Benutzer Zeit zur Eingabe hat, legen wir die Ausführung des Programms durch den Aufruf der Curses-Funktion `napms` für ein paar Millisekunden (`NAP_TIME`) auf Eis. Informationen zur Funktion `napms` finden Sie auf der Seite
http://invisible-island.net/ncurses/man/curs_kernel.3x.html
Gleich nach dem Aufwachen und als letzte Aktion in der Schleife zeigen wir alle Änderungen im virtuellen Fenster-Puffer an.

`@017`: Geben Sie den in Zeile 200 festgelegten Resultat-Code an den Aufrufer der Funktion `doLevel` zurück.

Schlußbemerkung

Nach der Vervollständigung des Codes versuchen Sie, das Programm `worm.c` durch Aufruf des Compilers `gcc` zu übersetzen und in ein ausführbares Programm zu überführen.

Das beigefügte Makefile hilft Ihnen bei dieser Aufgabe.

```
$ make clean
$ rm -rf bin

$ make
$ mkdir bin
$ gcc -g -Wall worm.c -o bin/worm -lncurses
```

Es ist von großer Wichtigkeit, dass Sie alle Teile des Programms verstehen. Studieren Sie insbesondere das Zusammenwirken der Funktionen und die eingesetzten Mechanismen der Parameterübergabe.
Nehmen Sie kleine Veränderungen am Code vor und verfolgen Sie deren Auswirkungen. Diskutieren Sie den Code mit Ihren Kommilitonen und Ihrem Betreuer im Praktikum.

Viel Spaß! `oooooooooo0`

Wichtiger Hinweis: Vergessen Sie nicht vor dem Herunterfahren der virtuellen Maschine Ihre in der virtuellen Maschine gemachten Änderungen Ihrem lokalen Repository hinzuzufügen.
Nützliche Kommandos: `git status`, `git add`, `git commit`
Das lokale Repository mit dem Repository bei Bitbucket zu synchronisieren:
Nützliche Kommandos: `git status`, `git push`