Technische Hochschule Ingolstadt
Fakultät für Informatik

Praktikum zu „Grundlagen der Programmierung“

Blatt 2 (Vorführaufgabe)

Lernziele: Erste Schritte mit der Curses-Bibliothek
Erforderliche Kenntnisse: Grundlegende Elemente und Aufbau von Programmen in der Sprache C

Voraussetzungen:

- Die Entwicklungsumgebung in der virtuellen Maschine ist im PC-Pool und wahlweise zusätzlich auf dem eigenen PC aktiviert und ausprobiert worden.
- Förderlich ist es, wenn Sie schon die einführenden Dokumente
  11_WichtigeShellKommandos.pdf
  12_ArbeitenMitEinemRevisionControlSystem.pdf
  40_EditCompileRundDebug.pdf
  50_Makefiles.pdf
  bearbeitet haben.
- Sie haben sich hinreichend mit dem Editor vim auseinander gesetzt (per Programm vimtutor oder per Online-Recherche) und können diesen flüssig bedienen.
- Im Folgenden gehen wir davon aus, dass Sie die Entwicklungsumgebung in der virtuellen Maschine ws23-AsmCpp2204-32bit oder eine ihrer Nachfolgeversionen im PC-Pool benutzen.

Die Vorführung von Aufgaben zur Erlangung von Testaten muss auf Basis dieser virtuellen Maschine erfolgen. Für die Vorführung muss zwingend ein Pool-Rechner im PC-Pool benutzt werden.

1 Einführung (Das Spiel Worm)

Im Rahmen des Praktikums programmieren wir das zeichenbasierte Spiel *Worm* für die Konsole. Worm ist eine Variante des Klassikers *Snake*. Anhand dieses Spiels werden alle, die Programmierung in der Sprache C betreffenden, Lehrinhalte der Vorlesung *Grundlagen der Programmierung I* vermittelt.

Abb. 1 zeigt das Spiel in der Endausbaustufe Worm100, die von Ihnen nach vollständiger Bearbeitung aller Praktikumsaufgaben erstellt wird. Der grüne Wurm ist der vom Benutzer gesteuerte Wurm. Die anderen Würmer werden automatisch bewegt und stehen in Konkurrenz zum grünen Wurm in ihrem Bestreben, alle Futterbrocken (Zahlen 2, 4, 6) möglichst schnell zu vertilgen, ohne dabei mit anderen Würmer und Hindernissen zu kollidieren. Der Bildschirmrand darf ebenfalls nicht berührt werden.
Nach dem Motto „Von 0 auf 100“werden wir diese Endversion, beginnend mit der ersten Version Wurm000, schrittweise erstellen und dabei in jeder weiteren Version Funktionalität und neue Konstrukte der Programmiersprache C hinzufügen.

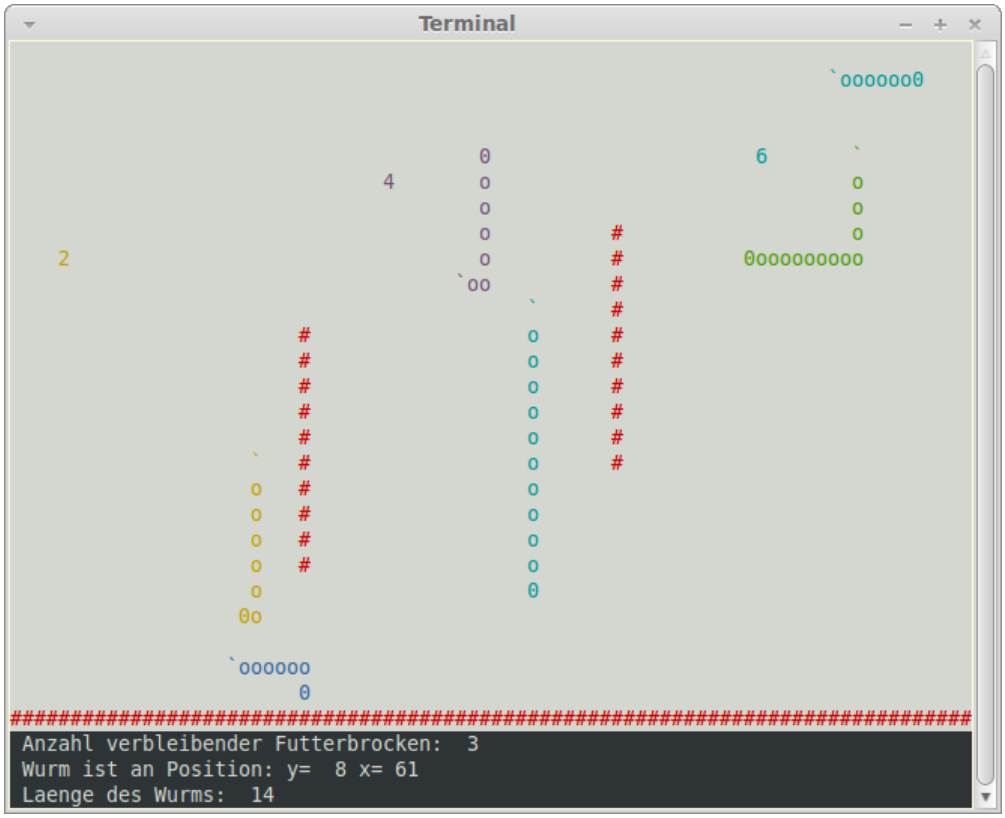

Abbildung 1: Endausbaustufe Worm100

2 Wieso Curses? (Grundsätzliches zur Curses-Bibliothek)

Die Ausgabe von Zeichen im Anzeigefenster eines Terminals (Konsole) an einer bestimmten Position auf der Basis von Koordinaten (Zeile, Spalte) kann nur mit sehr viel Mühe auf der Basis der C-Standardbibliotheken (stdio, stdlib) realisiert werden. Ebenfalls aufwändig ist die Realisierung einer nicht-blockierenden Abfrage von Tastatureingaben, z.B. die Prüfung, ob gerade die Taste X gedrückt wird.
Das Unterfangen wächst sich schnell zu einem ernsthaften Projekt aus, wenn die Lösung darüber hinaus plattformunabhängig sein soll, d.h. die Implementierung soll unverändert auf verschiedenen Betriebssystemen verwendbar sein, etwa unter Windows, MacOS und Unix.
Glücklicherweise gibt es bereits eine plattformunabhängige Implementierung der geforderten Funktionalität, nämlich in Form der Curses-Bibliothek. Die Bibliothek wird unter verschiedenen Namen für diverse Plattformen angeboten, jedoch mit einheitlicher Schnittstelle. Unter Windows unter dem Namen PDCurses (Public Domain Curses) und unter MacOS und Unix unter dem Namen NCurses (New Curses).
Die erste Version von Curses wurde bereits 1977 von Ken Arnold für BSD-Unix implementiert. Seither wurde sie mehrfach verbessert und auf viele Plattformen portiert. Sie stellt den de-facto Standard für textbasierte Benutzerschnittstellen (TUI: Text User Interface) dar und wird auch heute noch im Embedded-Bereich und für Spezialhardware zur Programmierung von Benutzerschnittstellen eingesetzt. Beispiele hierfür sind Storage-Systeme, Router, Oberflächen für BIOS-Programme und viele mehr. Link hierzu: http://en.wikipedia.org/wiki/Curses_(programming_library)

2.1 Dokumentation der Curses-Bibliohek im Netz

Die Curses-Bibliothek wird vorwiegend auf Unix-artigen Plattformen eingesetzt (NCurses), wohingegen der Port für Windows (PDCurses) einen Seitenast darstellt. Beide Linien werden jedoch aktiv gepflegt.
Aufgrund des Schwerpunkts im Unix-Umfeld beziehen sich die meisten Dokumente zu Curses auf die Variante NCurses. Sie bilden aber auch bei Verwendung der Variante PDCurses eine wertvolle Informationsquelle. Im Folgenden sind einige Links aufgeführt (alle Links zuletzt am 13.10.2019 geprüft):

http://tldp.org/HOWTO/NCURSES-Programming-HOWTO
: Eine Einführung in die Verwendung der Bibliothek NCurses

https://github.com/wmcbrine/PDCurses/blob/master/docs/README.md
: Eine ausführliche Einführung für PDCurses

http://invisible-island.net/ncurses/man/index.html
: Die Manual-Pages der Bibliothek. Auf einem Unix-System sind diese Manuale meistens auch installiert und können z.B. via man ncurses aufgerufen werden.

https://invisible-island.net/ncurses/hackguide.html
: Eine tiefergehende Dokumentation der Design-Ideen und Richtlinien für die Weiterentwicklung der Bibliothek NCurses.

Nutzen Sie die oben angegebenen Referenzen, insbesondere die erste, um sich mit den Grundzügen der Verwendung der Bibliothek vertraut zu machen.

3 Der erste Schritt (Worm000)

Im ersten Schritt (Version Worm000) machen wir uns vertraut mit der Ausgabe von Zeichen und Texten basierend auf Koordinaten (Zeile, Spalte) relativ zum Terminal-Fenster. Des Weiteren nutzen wir eine Tastaturabfrage, um die Anzeige erst zu löschen, wenn der Benutzer eine Taste drückt. Das Fenster, das durch den Start einer Shell dargestellt wird, bezeichnen wir im Folgenden als Terminal-Fenster.

3.1 Übersicht: Aufgabenbeschreibung für das gesamte Programm in der Version 000

Die einzelnen Teilaufgaben werden weiter unten in gesonderten Abschnitten genau erklärt:

- Schreiben Sie die Buchstaben A, B, C, D in die Ecken ihres Terminal-Fensters, so wie durch den folgenden Screenshot (Normaler Ablauf) dargestellt.
- Schreiben Sie sodann, mittig im Fenster zentriert, den Text
  `"Das Fenster hat yyy Zeilen und xxx Spalten"`
  Hierbei sollen die Platzhalter yyy und xxx durch die tatsächlichen Größen Ihres Fensters ersetzt werden (siehe Abb. 2: „normaler Ablauf“).

- Falls das Fenster weniger als 3 Zeilen hat oder zu schmal für die Darstellung des mittig zentrierten Textes ist, soll eine Fehlermeldung ausgegeben werden (siehe Abb. 3: „Fehlerfall“).
- In jedem Fall soll die Anzeige so lange stehen bleiben, bis der Benutzer eine Taste drückt. Erst danach darf die Anzeige gelöscht und das Programm beendet werden

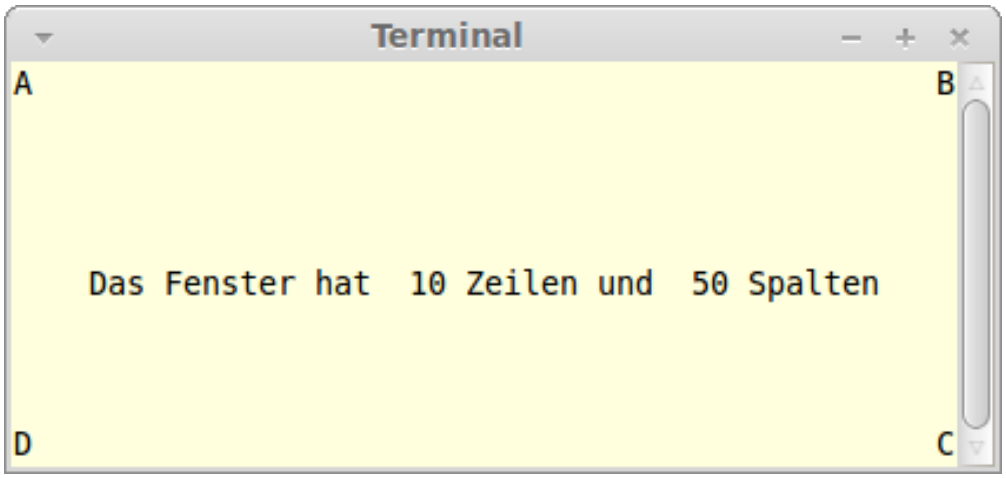

Abbildung 2: Normaler Ablauf

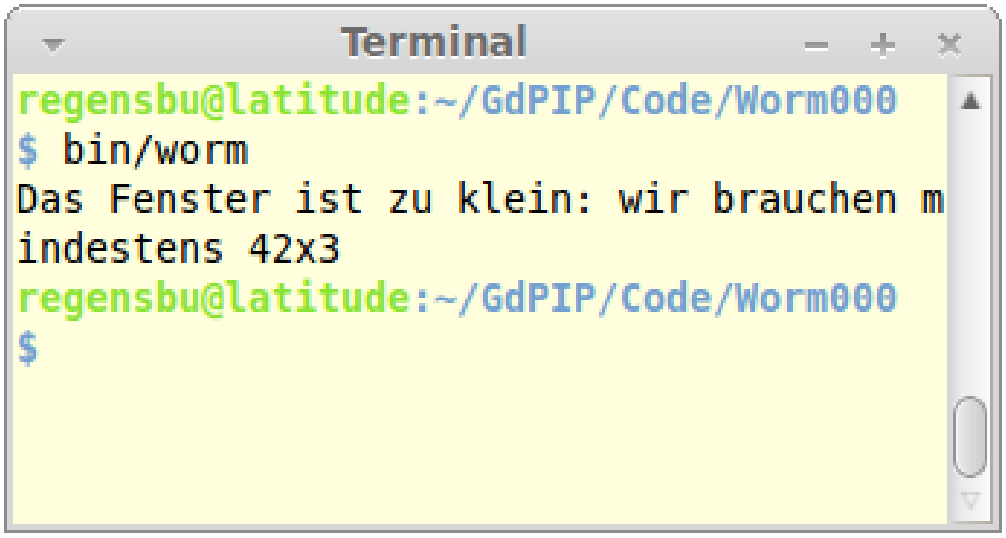

Abbildung 3: Fehlerfall

Aufgabe 1 (Kopieren, Auspacken und Umbenennen des Templates)

Holen Sie zunächst den aktuellen Datenbestand von Ihrem Repository bei bitbucket auf Ihre virtuelle Maschine.
An einem PC im PC-Pool im Kioskmodus: `git clone ...` *(siehe Blatt01)*
Zuhause mit privater VM im Normalmodus: `git pull`

Im Moodle-Kursraum finden Sie im Verzeichnis Praktikum die Datei `Worm000Template.zip`. Kopieren Sie diese Datei in das Verzeichnis `~/GdP1/Praktikum/Code` in Ihrer virtuellen Maschine. Öffnen Sie sodann eine Shell.

In der Shell wechseln Sie mit folgendem Befehl in das Verzeichnis `~/GdP1/Praktikum/Code`.
`$ cd ~/GdP1/Praktikum/Code`
Vergewissern Sie sich mittels des Befehls ls, dass die Zip-Datei im Verzeichnis vorhanden ist.
`$ ls Worm000Template.zip`
Entpacken Sie die Zip-Datei durch den nachfolgenden Befehl.
Dadurch wird das Verzeichnis `Worm000Template` mit einigen Dateien darin angelegt.
`$ unzip Worm000Template.zip`
Listen Sie den Inhalt des neuen Unterverzeichnisses.
`$ ls -la Worm000Template`
Kopieren Sie nun das Verzeichnis rekursiv in das Verzeichnis Worm000.
`$ cp -r Worm000Template Worm000`
Listen Sie den Inhalt des neuen Unterverzeichnisses.
`$ ls -la Worm000`
Listen Sie den Inhalt des aktuellen Verzeichnisses.
`$ ls -la`
Löschen Sie das Zip-Archiv.
`$ rm -f Worm000Template.zip`
Löschen Sie das Verzeichnis Worm000Template mit allen enthaltenen Dateien.
`$ rm -rf Worm000Template`
Listen Sie den Inhalt des aktuellen Verzeichnisses. `$ ls -la`

Hinweise: Die obigen Befehle sollen Sie mit der Manipulation von Archiven, Verzeichnissen und Dateien auf Shell-Ebene vertraut machen.

Die Generierung von zip-Archiven erfolgt einfach über das Programm `zip`.

Beispiel:
Erzeugung eines Archivs `Worm000.zip` mit Inhalt des Verzeichnisses `Worm000` Wechseln Sie in das Verzeichnis, das `Worm000` als Unterverzeichnis enthält. Dann
`$ zip -r Worm000 Worm000`
Im erzeugten Archiv `Worm000.zip` sind relative Pfade beginnend mit `Worm000` enthalten (Option `-r`).

Aufgabe 2 (Vervollständigung der Datei `worm.c`)

Öffnen Sie **zwei** Shells und wechseln Sie in jeder dieser Shells ins Verzeichnis `~/GdP1/Praktikum/Code/Worm000`.

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 nun hinzufügen sollen.
Die Form eines Lückentextes wurde hier gewählt, damit für Sie, als Einsteiger in die C-Programmierung, die Anfangshürde nicht zu hoch ist. Mit zunehmender Dauer des Praktikums werden Sie jedoch immer weniger Hilfestellung in Form von Code-Schablonen erhalten.
Öffnen Sie nun die Datei `worm.c` in einem Editor, damit wir unsere Tour durch die Datei beginnen können.

Zeile 11: Hier wird die Header-Datei `curses.h` eingebunden. Dadurch werden dem Compiler die Funktionen der Curses-Bibliothek bekannt gemacht.

Zeile 12: Wir binden die Header-Datei `string.h` ein, weil wir im Programm eine der String-Funktionen (Berechnung der Länge eines Strings) verwenden wollen. Dadurch werden dem Compiler die Funktionen der String-Bibliothek bekannt gemacht. Informationen zu den Funktionen der String-Bibliothek finden Sie im Netz unter

http://www2.hs-fulda.de/~klingebiel/c-stdlib/string.htm

Auf einem Unix-Rechner mit gepflegten Manual-Pages liefert der Befehl `man -s 3 string` entsprechendes.

Zeilen 19,20: Hier werden Konstanten für Resultat-Codes der Funktion `main()` definiert. Der Code 0 signalisiert im Allgemeinen die ordnungsgemäße Beendigung einer Funktion oder eines ganzen Programms. Ein Code verschieden von 0 signalisiert üblicherweise eine Fehlersituation. Aus Gründen der besseren Wartbarkeit werden symbolische Konstanten für die im Programm verwendeten Codes vereinbart.

Funktion `initializeCursesApplication()`: Diese Funktion kapselt mehrere Einstellungen, die wir für die Benutzung der Curses-Bibliothek in unserem Programm vornehmen wollen. Diese Hilfsfunktion wird uns durch das ganze Praktikum begleiten, da die vorgenommenen Einstellungen bei allen Versionen des Programms `worm` identisch sind.
Wir rufen die Funktion `initializeCursesApplication()` üblicherweise sehr früh in der Funktion `main()` auf.
Studieren Sie die Kommentare hinter den einzelnen Anweisungen innerhalb der Funktion.
Es handelt sich durchwegs um Aufrufe von Funktionen der Curses-Bibliothek.
Schlagen Sie die Dokumentation der einzelnen Funktionen in den folgenden Manualen nach:

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

http://www.tldp.org/HOWTO/NCURSES-Programming-HOWTO/

Achtung: Durch Aufruf der Funktion `initializeCursesApplication()` wird das Ein-Ausgabeverhalten des Terminal-Fensters, in dem unsere Curses-Anwendung läuft, stark verändert. Erst der Aufruf der Funktion `cleanupCursesApp()` überführt das Terminal wieder in einen normalen Zustand.

Funktion `cleanupCursesApp()`: Die Funktion ist das Gegenstück zur Funktion

`initializeCursesApplication()`. In ihr sind Einstellungen gekapselt, die das Terminal-Fenster, in dem unsere Curses-Anwendung läuft, wieder in einen normalen Zustand überführen. Diese Funktion muss zwingend vor Beendigung der Curses-Anwendung aufgerufen werden, auch im Fehlerfall, da sonst das Terminal für die weitere Nutzung in der Shell unbrauchbar wird.
Schlagen Sie die Dokumentation der in `cleanupCursesApp()` verwendeten Funktionen in den folgenden Manualen nach:

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

http://www.tldp.org/HOWTO/NCURSES-Programming-HOWTO/

Funktion `main()`: Die Hauptfunktion `main()` in der Datei `worm.c` realisiert die eingangs erwähnte Aufgabenstellung. Wir stellen in den Ecken die Buchstaben A - D dar und geben mittig zentriert einen Text aus.
In der Funktion `main()` sind die bereits erwähnten Lücken enthalten, die Sie ausfüllen sollen. Die Lücken sind mit Platzhaltern markiert, die alle die Form **`@nn`** haben, wobei `nn` eine Zahl ist. Wir benutzen diese Platzhalter, damit wir gezielter auf die Lücken im Programm Bezug nehmen können. Sie müssen diese Platzhalter durch entsprechende Anweisungen ersetzen.

`@01` Hier müssen Sie die Curses-Anwendung initialisieren. Die an dieser Stelle aufzurufende Funktion wurde oben bereits erwähnt.

`@02` Berechnen Sie durch den Aufruf einer geeigneten Funktion der String-Bibliothek (`string.h`) die Länge der Zeichenkette `message_template` und weisen Sie diese der Integer-Variable `msg_len` zu.
Hinweis: statt Verwendung einer Template-Zeichenkette könnte man auch eleganter die Funktion `sprintf()` benutzen. Die in diesem Zusammenhang notwendige Pufferverwaltung würde aber unsere erste Variante `Worm000` unnötig kompliziert machen.

`@03` Setzen Sie die Variablen `min_rows` und `msg_len` sinnvoll in der Bedingung ein. Die Makros `LINES` und `COLS` werden von der Curses-Bibliothek bereitgestellt. Sie liefern die Anzahl der Zeilen und Spalten des Terminal-Fensters.

`@04` Wir befinden uns in einem Fehlerzweig und möchten eine Fehlermeldung mit konventionellen Mitteln der C-Standardbibliothek (`printf`) ausgeben. Zuvor müssen wir jedoch das Terminal-Fenster wieder in einen normalen Zustand überführen. Die an dieser Stelle aufzurufende Funktion wurde oben bereits erwähnt.

`@05` Formatierte Ausgabe mittels `printf()`. Setzen die Variablen `msg_len` und `min_rows` sinnvoll ein.

@06 Berechnen Sie unter Verwendung der Makros `LINES` und `COLS` die Position der mittleren Zeile und eine Anfangsspalte, so dass der Text im Fenster zentriert ausgegeben wird. Speichern Sie die Werte in den Variablen `mid_row` und `start_col`.

@07 Positionieren Sie die Schreibmarke in der linken oberen Ecke des Terminal-Fensters und geben Sie dort den Buchstaben A aus. Nutzen Sie hierzu die Funktionen `move()` und `addch()`. Schlagen Sie die Dokumentation dieser Funktionen in einem Curses-Manual nach. (z.B. http://invisible-island.net/ncurses/man/ncurses.3x.html)

@08,@09,@10 Positionieren Sie die Schreibmarke in die anderen drei Ecken und geben Sie den entsprechenden Buchstaben dort aus. Nutzen Sie hierzu die kombinierte Funktion `mvaddch()`. Schlagen Sie die Dokumentation dieser Funktion in einem Curses-Manual nach.

@11 Setzen Sie die beiden Makros `LINES` und `COLS` sinnvoll ein, um die aktuellen Dimensionen Ihres Fensters auszugeben. Schlagen Sie die Bedeutung der Makros in einem Curses-Manual nach.

@12 Damit die von Ihnen vorgenommen Änderungen an den Ausgabe-Puffern tatsächlich sichtbar werden muss die Funktion `refresh()` aufgerufen werden. Schlagen Sie deren Bedeutung im Curses-Manual nach.

@13 In der nächsten Zeile rufen wir die Funktion `getch()` auf, welche die Tastatur abfragt. Entgegen unserer üblichen Präferenz, dass die Tastaturabfrage während des Spiels nicht blockieren soll, möchten wir jetzt, dass das Programm in der Funktion `getch()` solange wartet, bis eine beliebige Taste gedrückt wird.
Zu diesem Zweck müssen wir das Verhalten der Eingabefunktion `getch()` ändern. In der Funktion `initializeCursesApplication()` stellen wir üblicherweise ein, dass `getch()` nicht blockiert.
Stellen Sie nun durch den Aufruf einer geeigneten Bibliotheksfunktion sicher, dass der nächste Aufruf von `getch()` blockiert bis eine Taste gedrückt wird.

@14 Rufen Sie unsere Hilfsfunktion `cleanupCursesApp()` auf, um dem Terminal-Fenster wieder ein normales Ein-/Ausgabeverhalten zu geben.

Damit sind wir am Ende der Vervollständigung des Templates für das Programm `worm.c`. Durchlaufen Sie nun solange den Mikrozyklus `Edit`/Compile/`Run` bis Ihre Anwendung fehlerfrei übersetzt wird und das geforderte Verhalten zeigt.

Sie können das ausführbare Programm `bin/worm`
durch eine der beiden Varianten erzeugen:

1. `mkdir -p bin; gcc -g -Wall worm.c -o bin/worm -lncurses`

2. `make clean; make`

Abnahme der Vorführaufgabe

Ihr Kursbetreuer wird sich von Ihnen zum Zweck der Lern- und Erfolgskontrolle das Programm vorführen lassen. Rechnen Sie damit, dass Sie

- Ihrem Betreuer den Zweck einzelner Anweisungen im Programm erklären müssen
- auf Anfrage individuelle Änderungen vornehmen und erklären müssen
- zeigen müssen, dass Sie den Mikrozyklus `Edit/Compile/Run` beherrschen
- Ihren C-Code sauber formatiert haben (einheitlich Einrücken) und an geeigneten Stellen Ihren Code auch sinnvoll dokumentieren
- in der Lage sind, den C-Code in den dafür vorgesehenen Unterverzeichnissen ordentlich zu organisieren.
- in der Lage sind, ihr `git`-Repository bei *bitbucket* zu verwalten und den sicheren Umgang mit den Befehlen `git status/add/commit/clone/push/pull` beherrschen.

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`