Studiengänge IB, FFI, CSI
WS 2023/24

Technische Hochschule Ingolstadt
Fakultät für Informatik

Praktikum zu „Grundlagen der Programmierung“

Blatt 4 (Vorführaufgabe)

Lernziele: Verwendung der Datentypen `enum` und `bool`
Erforderliche Kenntnisse: keine besonderen
Voraussetzungen:

- Vollständige Bearbeitung des letzten Blattes 03 (`Worm005`).

Übersicht

Im Rahmen dieses Aufgabenblatts führen wir kleine Verbesserungen an der letzten Version Worm005 durch, die der besseren Lesbarkeit unseres Programms dienen.
Eine der wichtigsten Lektionen im Rahmen der Ausbildung zum Informatiker ist folgende:

> **Unsere Programme (Software-Systeme) müssen nicht nur funktionieren, sondern sie müssen von anderen verstanden und gewartet werden können.**

Wir schreiben also unsere Programme nicht nur so, dass der Compiler sie akzeptiert, sondern fügen aussagekräftige Kommentare hinzu und bemühen uns um eine klare Struktur, so dass andere Entwickler, die später unseren Code warten müssen, verstehen, was wir uns gedacht haben.
Im Rahmen unseres Praktikums verwenden wir lediglich die in die Sprache C eingebauten Mechanismen zur Kennzeichnung von Kommentaren (`//` und `/* ... */`). Im professionellen Entwicklungsumfeld geht man weit darüber hinaus und setzt etwa Dokumentationsgeneratoren wie *Doxygen* (https://www.doxygen.org) ein.
Für einige Programmiersprachen gibt es auch speziell auf die Sprache zugeschnittene Werkzeuge zur Erzeugung von Dokumentation, z.B. *javadoc* für die Sprache *Java* (https://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html).
Zu einer klaren Struktur gehört auch, dass wir bei Funktionsparametern und Funktionsresultaten die zu verwendenden Datentypen so restriktiv wie möglich spezifizieren. Aus historischen Gründen werden gerade in der Sprache C die ganzzahligen Datentypen (**`char`** , **`short`**, **`int`**, ...) oft zur Kodierung von Aufzählungen und Wahrheitswerten verwendet.
Sollen Elemente von Aufzählungen oder Wahrheitswerte als Funktionsargumente übergeben oder als Funktionsresultate zurück geliefert werden, so ist diese Intention des Entwicklers oft nicht direkt aus der Signatur der Funktionen ersichtlich. Dort finden sich nur die Typen **`char`**, **`short`**, **`int`**, die zur Kodierung der Aufzählung oder der Wahrheitswerte verwendet wurden. Allzu leicht kann es passieren, dass der unwissende Benutzer einer Funktion beliebige Zahlen als Argumente übergibt und nicht nur solche, die bestimmte Konstanten kodieren.
An dieser Stelle helfen die in den neueren C-Standards eingeführten Datentypen für Enumerationen **`enum`** (C90) und der spezielle Datentyp für Wahrheitswerte **`bool`** (C99).
Im Rahmen dieses Aufgabenblatts wird es Ihre Aufgabe sein, die Version `Worm005` in die Version `Worm010` zu überführen und dabei an allen Stellen, wo es sinnvoll ist, Kodierungen durch den Datentyp **`int`** durch spezifischere Kodierungen in Form von Enumerationen (**`enum`**) und Wahrheitswerten (**`bool`**) zu ersetzen.

Aufgabe 1 der Version **Worm005** nach **Worm010**

Anstatt unsere Version `Worm005` abzuändern, stellen wir eine Kopie in `Worm010` her, damit wir auf der Kopie arbeiten können.
Wechseln Sie in einer Shell in das Verzeichnis `~/GdP1/Praktikum/Code`:
`$ cd ~/GdP1/Praktikum/Code`
Mit folgendem Befehl stellen wir eine Kopie des Verzeichnisses `Worm005` in einem neuen Verzeichnis `Worm010` her. Die Option `-r` steht für rekursives Kopieren.
`$ cp -r Worm005 Worm010`
Wechseln Sie in das neue Verzeichnis `Worm010`.
`$ cd Worm010`
Mit einem abschließenden `make clean` räumen wir alte Kompilate im Verzeichnis bin auf.
`$ make clean`
Ändern Sie nun noch die Datei `Readme.fabr` ab, indem Sie folgenden Inhalt eintragen:

```
New in this version
Enhanced data types
- use enum
- use bool
```

Aufgabe 2 (1 Ersetze **CPP**-Konstanten **TRUE**/**FALSE** durch **true**/**false** des Typs **bool**)

Im Standard C99 wird der Datentyp **bool** durch die Bibliothek `stdbool` eingeführt. Durch Einfügen (*#include*) der Header-Datei `stdbool.h` hat man Zugriff auf die beiden Elemente `true` und `false` des neuen Typs **bool**.
Technisch werden der Datentyp **bool** und seine Elemente `true`/`false`, wie bisher auch, durch Definitionen des C-Präprozessors (`CPP`) realisiert. Jedoch führt die Verwendung des Datentyps `bool` zu besserer Lesbarkeit und Standardisierung der Bezeichner.
Öffnen Sie nun im Verzeichnis `Worm010` die Datei worm.c in einem Editor. Fügen Sie nach der `CPP`-Anweisung

```
#include <stdlib.h>
```

die Anweisung

```
#include <stdbool.h>
```

ein. Durch das Inkludieren der Datei `stdbool.h` sind der Datentyp **bool** und seine Elemente `true`/`false` ab dieser Stelle definiert.
Nun müssen wir alle Stellen im Programm `worm.c` finden, an denen die alten `CPP`-Konstanten `TRUE` und `FALSE` eingeführt bzw. benutzt wurden, und dort die neuen Konstanten `true` und `false` einsetzen.
Zunächst löschen wir die Definitionen (*#define*) der alten `CPP`-Konstanten `TRUE` und `FALSE`.
Danach nutzen wir den Editor, um (fast) alle Vorkommen der alten Konstanten `TRUE` und `FALSE` durch die neuen Bezeichner `true` und `false` zu ersetzen. Wir dürfen dabei jedoch solche Vorkommen, die ‚alte' Schnittstellen bedienen, nicht ersetzen. Zu solchen Vorkommen alter Schnittstellen

zählen die Aufrufe der Curses-Funktionen `keypad` und `nodelay`, denn die Bibliothek Curses setzt den neuen Datentyp **`bool`** offensichtlich nicht ein. Deshalb definiert die Header-Datei `curses.h` die Konstanten `TRUE` und `FALSE`.
Im Editor `vim` erledigt das interaktive Suchen und Ersetzen das Kommando `%s` (siehe `vimtutor`). So sucht z.B.
`:%s/TRUE/true/gc`
interaktiv nach `TRUE` und fragt bei jedem Vorkommen, ob ersetzt werden soll.
Nach `TRUE` ersetzen wir analog die Konstante `FALSE` durch die neue Konstante `false`, wobei wir auch hier die Argumente der Funktion `nodelay` aussparen.
Als letztes prüfen wir, ob wir den Datentyp **`bool`** bei Definitionen von Variablen oder Funktionen einsetzen können.
Im Programm `worm.c` in der aktuellen Version ist die diesbezügliche Ausbeute aber gering. Lediglich die Funktion `doLevel` enthält die Variable `end_level_loop`, die bisher als Integer-Variable vom Typ `int` definiert wird. Hier ändern wir den Typ und definieren `end_level_loop` als Variable vom Typ **`bool`**.
Aus stilistischen Gründen können wir dann noch in der **`while`**-Schleife der Funktion `doLevel` den Vergleich `end_level_loop == false` in den eleganteren Ausdruck `!end_level_loop` umwandeln.

Aufgabe 3 (Einführung der Enumeration `ResCodes`)

Zu Beginn der Datei `worm.c` werden bisher folgende Konstanten definiert:

```
// Result codes of functions
#define RES_OK 0
#define RES_FAILED 1
```

Ersetzen Sie diese `CPP`-Definitionen durch folgende Enumeration:

```
// Result codes of functions
enum ResCodes {
  RES_OK,
  RES_FAILED,
};
```

Das abschließende Komma `,` vor der schließenden Klammer } ist kein Syntaxfehler, sondern ist ausdrücklich im Sprachstandard erlaubt. Es erleichtert ein späteres Hinzufügen weiterer Konstanten.
Nachdem wir den neuen Typ **`enum`** `ResCodes` eingeführt haben, suchen wir jetzt alle Stellen, an denen wir statt des nichtssagenden Typs **`int`** den spezifischeren Typ **`enum`** `ResCodes` einsetzen können. An folgenden Stellen können wir den neuen Typ einsetzen (nutzen Sie die Suchfunktion des Editors):

- Resultat-Typ der Funktion `doLevel`
- In Folge für Variablen, die das Ergebnis von `doLevel` speichern
- Resultat-Typ der Funktion `initializeWorm`
- In Folge für Variablen, die das Ergebnis von `initializeWorm` speichern

Hinweis: eine konsequente Fortführung dieses Gedankengangs führt dazu, dass wir als Resultat-Typ der Funktion `main` den Typ **`enum`** `ResCodes` angeben möchten. Wir belassen den Typ von `main` jedoch bei **`int`**, da wir andernfalls eine Warnung des C-Compilers ernten.
Dieser mag es lieber, wenn `main` ein **`int`**-Resultat liefert.

Aufgabe 4 (Einführung der Enumeration `ColorPairs`)

Zu Beginn der Datei `worm.c` wird bisher folgende Konstante definiert:

```
// Numbers for color pairs used by curses macro COLOR_PAIR
#define COLP_USER_WORM 1
```

Ersetzen Sie diese CPP-Definitionen durch folgende Enumeration

```
// Numbers for color pairs used by curses macro COLOR_PAIR
enum ColorPairs {
  COLP_USER_WORM = 1,
};
```

Hier sehen wir ein Beispiel, wie wir die interne Kodierung der Enumeration durch den C-Compiler beeinflussen können. Wir erzwingen, dass die Konstante `COLP_USER_WORM` in der Enumeration den Code 1 bekommt. Später, wenn wir automatische Systemwürmer mit zufälligen Farben programmieren, werden wir ausnützen, dass das Farbpaar mit dem Code 1 für Systemwürmer verboten ist. Diese Paarung ist dem Benutzerwurm vorbehalten. Nachdem wir den neuen Typ **`enum`** `ColorPairs` eingeführt haben, suchen wir jetzt alle Stellen, an denen wir statt des nichtssagenden Typs **`int`** den spezifischeren Typ **`enum`** `ColorPairs` einsetzen können. An folgenden Stellen können wir den neuen Typ einsetzen (nutzen Sie die Suchfunktion des Editors):

- Typ der globalen Variablen `theworm_wcolor`
- Parameter `color_pair` der Funktion `placeItem`
- Parameter `color` der Funktion `initializeWorm`

Aufgabe 5 (Einführung der Enumeration `GameStates`)

Zu Beginn der Datei `worm.c` werden bisher folgende Konstanten definiert:

```
// Game state codes
#define  WORM_GAME_ONGOING  0
#define  WORM_OUT_OF_BOUNDS 1   // Left screen
#define  WORM_GAME_QUIT     2   // User likes to quit
```

Ersetzen Sie diese CPP-Definitionen durch folgende Enumeration

```
// Game state codes
enum GameStates {
  WORM_GAME_ONGOING,
  WORM_OUT_OF_BOUNDS,    // Left screen
  WORM_GAME_QUIT,        // User likes to quit
};
```

Nachdem wir den neuen Typ **enum** `GameStates` eingeführt haben, suchen wir jetzt alle Stellen, an denen wir statt des nichtssagenden Typs **int** den spezifischeren Typ **enum** `GameStates` einsetzen können. An folgenden Stellen können wir den neuen Typ einsetzen (nutzen Sie die Suchfunktion des Editors):

- Parameter `agame_state` der Funktion `readUserInput`
- In Folge für Variablen, die als Argument auf dieser Parameterposition an `readUserInput` übergeben werden. Dies betrifft die Variable `game_state` der Funktion `doLevel`.
- Parameter `agame_state` der Funktion `moveWorm`
- In Folge für Variablen, die als Argument auf dieser Parameterposition an `moveWorm` übergeben werden. Dies betrifft wieder die Variable `game_state` der Funktion `doLevel`.

Aufgabe 6 (Einführung der Enumeration **WormHeading**)

Zu Beginn der Datei `worm.c` werden bisher folgende Konstanten definiert:

```
// Directions for the worm
#define WORM_UP      0
#define WORM_DOWN    1
#define WORM_LEFT    2
#define WORM_RIGHT   3
```

Ersetzen Sie diese `CPP`-Definitionen durch folgende Enumeration

```
// Directions for the worm
enum WormHeading {
  WORM_UP,
  WORM_DOWN,
  WORM_LEFT,
  WORM_RIGHT,
};
```

Nachdem wir den neuen Typ **enum** `WormHeading` eingeführt haben, suchen wir jetzt alle Stellen, an denen wir statt des nichtssagenden Typs **int** den spezifischeren Typ **enum** `WormHeading` einsetzen können.
An folgenden Stellen können wir den neuen Typ einsetzen (nutzen Sie die Suchfunktion des Editors):

- Parameter `dir` der Funktion `initializeWorm`
- Parameter `dir` der Funktion `setWormHeading`

Hinweis: Die auf den entsprechenden Parameterpositionen übergebenen Argumente sind immer bereits Konstante des Aufzählungstyps **enum** `WormHeading`. Es müssen daher keine Typen von Variablen geändert werden.

Schlußbemerkung

Nach der obigen Umstellung des Codes versuchen Sie bitte, das Programm `worm.c` durch Aufruf des Compilers `gcc` zu übersetzen und in ein ausführbares Programm zu überführen.
Das beigefügte Makefile hilft Ihnen wieder bei dieser Aufgabe.

```
$ make clean; make
```

Da wir nur Datentypen spezifischer gemacht haben, sollte das Programm `Worm010/bin/worm` das gleiche Verhalten zeigen wie sein Vorgänger `Worm005/bin/worm`.
Zum Schluss verschaffen Sie sich bitte mit Hilfe eines *Diff*-Tools einen Überblick, welche Änderungen sich durch den Einsatz des Datentyps **`bool`** und der neu eingeführten Enumerationen `ResCodes`, `ColorPairs`, `GameStates` und `WormHeading` ergeben haben.
Ein gutes *Diff*-Tool, welches für viele Plattformen verfügbar ist, ist das Programm `kdiff3`.
https://kdiff3.sourceforge.net/
Mit seiner Hilfe können Sie die Dateien der Verzeichnisse `Worm005` und `Worm010` rekursiv vergleichen.
In der von uns verwendeten virtuellen Maschine ist das Werkzeug `kdiff3` bereits vorinstalliert.
Hinweis: Installiert ist das Paket `kdiff3-qt`, wo `kdiff3` direkt gegen die QT-Bibliothek gelinkt ist, im Gegensatz zum normalen Paket `kdiff3`, das gegen die weitaus umfangreichere *KDE*-Bibliothek gelinkt ist.
Unter Windows wird `kdiff3` mit einem bequemen Plugin für den Explorer installiert.

Den Vergleich starten Sie unter Linux durch den Aufruf von `kdiff3` im Verzeichnis Code, welches die Unterverzeichnisse `Worm005` und `Wom010` enthält. Bitte denken Sie daran, vor dem Aufruf in beiden Verzeichnissen ein `make clean` auszuführen.
Der Vergleich von Binärdateien ist in diesem Zusammenhang nicht sonderlich hilfreich.

```
$ kdiff3 Worm005 Worm010 &
```

Sie werden feststellen, dass das Programm `worm.c` in der Version `Worm010` im Vergleich zur Version `Worm005` besser lesbar geworden ist. Insbesondere sieht man schneller und leichter, welchen Zweck die von der Umstellung betroffen Funktionsparameter haben.

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`