Hochschule RheinMain
Fachbereich DCSM - Informatik
Marcus Thoss, M.Sc.
Prof. Dr. Robert Kaiser
Christian Neugebauer

Hardwarenahe Programmierung I

WS 2021/22
LV 1512

Abgabe 2

Zur Berücksichtigung der Leistung muss eine Zip-Datei mit dem Dateinamen

`Nachname_Vorname_hwp1_21_a02.zip`

bis zum 03.02.2022 23:59 Uhr per E-Mail an `christian.neugebauer@hs-rm.de` gesendet werden. In der Zip-Datei muss Ihr C-Projekt enthalten sein. Im C-Projekt sind nur Quelltextdateien und das Makefile.

Teil der Abgabe ist ein sauber dokumentierter Code. Bitte dokumentieren Sie jede von Ihnen implementierte Funktionen und Module auf Englisch. Inhalt sollten Eingabe und Ausgabe Parameter und eine kurze Zusammenfassung der Funktion (bzw. der Module) sein.

Vergewissern Sie sich, dass das Display bündig aufgesteckt ist und mit dem Testprogramm des letzten Aufgabenblattes zumindest leuchten konnte.

Aufgabe 2.1 (Display-Test (5 P.)):

a) Legen Sie ein neues Programmverzeichnis für dieses Blatt an.

   Laden Sie von der Veranstaltungshomepage die Dateien `dprintf.h`, `libdprintf.a` `dprintftest.c` und `dprintftest` in das neue Verzeichnis herunter.

b) Laden Sie die vorgefertigte Datei `dprintftest` mit `avrdude` wie bei den Übungen von Blatt 5 auf Ihr Mikrocontroller-Board. Sie sollten die Ausgabe `Hello` sehen sowie eine Zahl, die pro Sekunde erhöht wird und ein einzelnes Zeichen, das sich beim Betätigen der Taster verändert.

c) Sehen Sie sich das Programm `dprintftest.c` und die Datei `dprintf.h` im Quelltext an, um sie in allen Details zu verstehen.

d) Schreiben Sie ein Makefile, mit dem Sie selbst `dprintftest` aus den Quellen erzeugen können.

   Hinweis:

Bei `libdprintf.a` handelt es sich um eine *Bibliothek* (*Library*), in der mehrere Objektdateien (`.o`) zusammengebunden sind.[1]

Für den *Linker*schritt müssen Sie daher neben `dprintftest.o` auch diese Bibliothek hinzulinken. Geben Sie dafür als Option beim Linken `-L.` an (beachten Sie den Punkt!), damit die Bibliothek im aktuellen Verzeichnis gefunden werden kann. Den Dateinamen der Bibliothek müssen Sie natürlich auch angeben, wie Sie es bei eine Objektdatei tun.[2]

e) Testen Sie nun mit dem von Ihnen erzeugten `dprintftest` weiter.

f) Nutzen Sie die mit `#if 0 ... #endif` im Quellcode deaktivierten Passagen, um `dprintftest.c` zu erweitern und testen Sie die zusätzlichen Features der Bibliothek. Kurze Beschreibungen der möglichen Funktionen finden Sie (natürlich) in `dprintf.h`

Aufgabe 2.2 (Ziffernraten auf dem Arduino (20P.)):

a) Konzipieren und realisieren Sie eine Variante Ihrer Lösung zu Aufgabe 2.1, die mit dem Display und seinen Tasten auf dem Mikrocontroller-Board funktioniert

b) Die Eingabe kann nun natürlich nicht auf einer „normalen“ Tastatur erfolgen. Überlegen Sie stattdessen, wie Sie eine Zahl eingeben lassen können, z.B. indem Sie das Steuerkreuz für das Verändern des Wertes um ± 1 verwenden lassen.

Aufgabe 2.3 (Blackjack mit Arduino und Display (25P.)):

Sie portieren Ihre Umsetzung des Kartenspiels Blackjack von Blatt 4 nun auf den Arduino Leonardo. Zur Anzeige wird das 16x2 Zeichen große LC-Display verwendet.

a) Legen Sie ein neues Programmverzeichnis für dieses Blatt an und kopieren Sie die C-Quelldateien Ihrer bisherigen Blackjack-Implementierung dort hinein.

b) Nutzen Sie die aus den bisherigen Aufgaben bekannten Dateien `dprintf.h`, `libdprintf.a` und kopieren Sie sie in das neue Verzeichnis.

c) Legen Sie die Dateien `user_led.h` und `user_led.c` an und kopieren Sie dorthin die LED-Funktionen aus Aufgabenblatt 5. Sorgen Sie dafür, dass die Deklarationen der Funktionen mit geeigneter Dokumentation in `user_led.h` stehen und die Datei passende Header-Guards (`#ifdef...`) erhält.

d) Ändern Sie `blackjack.c` so, dass sie für Arduino-Pragrammierung angepasst ist und einen rudimentären Display-Test enthält. Den restlichen Quellcode in `main()` können Sie zunächst auskommentieren und Schritt für Schritt wieder aktivieren.

e) Erstellen Sie ein `Makefile`, das alle Quellcode-Dateien compiliert und das AVR-Programm für Blackjack erstellt.

f) *Optional*: Nutzen Sie `avr-ar`, um aus den Objektdateien für Sound und LED eine eigene Bibliothek nach dem Vorbild von `libdprintf.a` zu erzeugen und diese beim Linkschritt statt der einzelnen `.o`-Dateien zu verwenden. `avr-ar`[3] ist dem `tar`-Kommando in vielerlei Hinsicht

[1] Sie können sich den Inhalt der Bibliothek mit dem Tool `avr-ar` ansehen. Finden Sie mit `man ar` heraus, wie!

[2] Alternativ können Sie auch `-ldprintf` *am Ende* des Linker-Aufrufs angeben. Diese Variante wird zum Ende der Vorlesung noch einmal betrachtet.

[3] `avr-ar` besitzt keine eigene `man`-Page, nutzen Sie `man ar`

ähnlich

g) Versionieren Sie alle Quelldateien und das `Makefile` in GitLab.

h) Wenn Ihr erster Test funktioniert, beginnen Sie mit der *Portierung* Ihres Blackjack-Codes auf die neue Zielplattform Arduino Leonardo. Die Ausgaben sollen statt auf der Shell nun auf dem Display erfolgen, die Eingaben über die Taster.

i) Versuchen Sie zunächst den für `test_print_player()` benötigten Code anzupassen. Hierzu muss `ui.c` so angepasst werden, dass `printf()` durch eine Kombination der verschiedenen `dprintf...()`-Funktionen der `dprintf`-Bibliothek ersetzt werden. Erstellen Sie auf diese Weise ein Blackjack-Programm, das zunächst nur `test_print_player()` mit Display-Ausgabe aufruft.

   Ein Vorschlag zur Gestaltung des Displays ist die Ausgabe der Karten auf der Spielerhand mit jeweils zwei Zeichen pro Karte (z.B. `H9` für Herz 9) in der ersten Zeile und eine Statusmeldung/Eingabeaufforderung in der zweiten Zeile

   Hinweis: Benutzen Sie auch Ihre Funktionen für die Ansteuerung der User-LED aus den vorherigen Aufgaben zum Debugging.

j) Auf eine Eingabe des Spielernamens sollten Sie in der Arduino-Version verzichten. Für die Spielzüge wie *Hit* und *Stay* überlegen Sie sich passende Abbildungen auf die Eingabetasten und setzen diese im Code um.

   Hinweise:

   - Die über `stdlib.h` verfügbaren Funktionen sind auf der AVR-Plattform eingeschränkt und in `man avr_stdlib` dokumentiert.
   - Wenn Sie `srandom()` mit einem Timer-basierten, also quasi-zufälligen Seed-Werte aufrufen wollen, lässt sich das nicht mehr über `time()` realisieren: verwenden Sie stattdessen den Zählerstand der Timers 0 im ATmega32U4. Sie erhalten diesen als 8-BitWert über das Register `TCNT0`, das Sie wie eine globale Variable auslesen können (via <avr/io.h> deklariert).

k) Die nächste Erweiterung Ihrer Lösung ist der Umbau der Datenmodellierung. Führen Sie dazu einen `struct Card`-Typ für eine Spielkarte in `game.h` als Deklaration ein. Diese `struct` soll dann zukünftig statt getrennter Werte für Wert und Farbe durchgängig im Code verwendet werden. Darin sollte dann die Farbe auch nicht mehr über Zahlenkonstanten, sondern über ein `enum Colour` abgebildet werden.

   Der Umbau Ihres restlichen Codes auf die Verwendung der neuen Datenstruktur ist eine Maßnahme, die in die Kategorie *Refactoring* fällt. Es ist leider auch in der Praxis nicht unüblich, dass sich ein Design, z.B. aufgrund veränderter Anforderungen, als nicht mehr angemessen oder ausreichend erweist und durch eine neues Design mit veränderten Schnittstellen und Datentypen ersetzt werden muss.