Übungsblatt 2

Dr. Matthias Frank, Dr. Matthias Wübbeling

Ausgabe Mittwoch, 18. Oktober 2023
Abgabe bis **Freitag, 27. Oktober 2023, 23:59 Uhr**
Vorführung vom 30. Oktober bis zum 3. November 2023

Alle Programme müssen unter **Ubuntu 22.04** kompilierbar bzw. lauffähig und ausreichend kommentiert und mit **Makefile** (C, Assembler) versehen sein, um Punkte zu erhalten. Als Compiler sollen **clang** (C) und **nasm** (Assembler) verwendet werden. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Alle Gruppenmitglieder sollten die Abgabe erklären können. Die Abgabe erfolgt mittels Ihres Git-Repositories und der vorgegebenen Ordnerstruktur.

Die Punkte der Aufgaben sind relevant für die Zulassung. Die Punkte der Bonusaufgaben werden auf Ihren Punktestand addiert, werden aber nicht auf die für die Zulassung benötigten Punkte addiert.

Abgabestruktur: Jede Abgabe, die die folgende Struktur nicht umsetzt, wird **NICHT** bepunktet bzw. mit 0 Punkten bewertet

- die zu korrigierenden Lösungen müssen bis zur Deadline auf dem `master`-Branch liegen. Lösungen auf anderen Branches werden nicht gewertet
- alle Lösungen müssen in der vorgegebenen Ordnerstruktur (**blattXX/aufgabeYY**) abgelegt werden, wobei `XX` und `YY` durch die jeweiligen Nummern des Zettels und der Aufgaben ersetzt werden sollen. Der Name soll exakt nur aus diesen Zeichen bestehen und achtet auf Kleinschreibung
- sämtliche Aufgaben, mit Ausnahme der theoretischen Aufgaben, die eine PDF erfordern, benötigen zwingend ein **Makefile**. Abgaben ohne dieses werden nicht gewertet

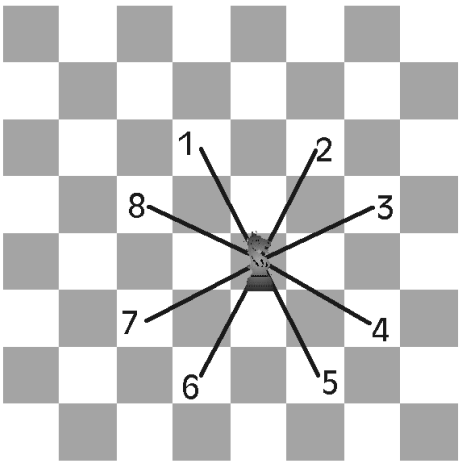
Hinweis: Die Vorführungen von Mittwoch, 1. November 2023, werden in der darauffolgenden Woche nachgeholt

Aufgabe 1 Das Springerproblem (4 Punkte)

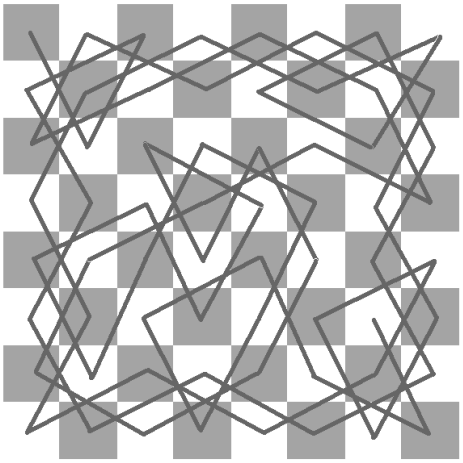
Es gibt manchmal Probleme, die so schwer sind, dass sie entweder nur Verrückten oder Genies einfallen konnten. Und dann gibt es auch wieder solche wie das Springerproblem.

Ausgedacht wurde es vor langer Zeit, vor über 200 Jahren, genauer im Jahre 1758, und zwar von einem Schweizer. Dieser Schweizer war kein geringerer als Leonhard Euler, der in Berlin saß und eines schönen Abends nicht an diese damals wie heute zu tiefst provinzielle Stadt dachte, sondern an etwas größeres, an ein Schachbrett:

> „Gegeben sei ein leeres Schachbrett. Gibt es eine Zugfolge, mit der der Springer alle (schwarzen und weißen) Felder des Brettes genau einmal besucht?“
>
> – (Quelle: `http://www.axel-conrad.de/springer/springer.html`)



(a) Erlaubte Züge eines Springers



(b) Eine Lösung für ein 8×8-Brett

Gegeben sei nun folgendes ANSI C-Programm zur Bestimmung aller Lösungen des Springerproblems, welches nach dem Prinzip des Backtracking arbeitet:

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

#include "chessboard.h"

int n = 5;
int start_x = 0;
int start_y = 0;
int solcnt = 0; // solution counter
int moves[8][2] = {{2,1}, {1,2}, {-2,1}, {-1,2}, {2,-1}, {1,-2}, {-2,-1}, {-1,-2}};
struct board_t b;

void knights_tour(struct board_t* b);

int main() {
        if (init_board(&b, n, start_x, start_y) > 0)
                return EXIT_FAILURE;

        knights_tour(&b);
        free_board(&b);
        return EXIT_SUCCESS;
}

void knights_tour(struct board_t* b) {
        if (visited_fields(b) >= n*n) {
                printf("Solution %i:\n", ++solcnt);
                print_board(b);
                printf("\n");
        } else {
                int i;
                for (i=0; i<8; i++) {
                        if (isfree(b, moves[i][0], moves[i][1])) {
                                new_jump(b, moves[i][0], moves[i][1]);
                                knights_tour(b);
                                remove_jump(b, moves[i][0], moves[i][1]);
                        }
                }
        }
}
```

Das angegebene Programm `springerproblem.c` ist in die PDF-Datei eingebettet.

Entwickeln Sie eine Bibliothek `chessboard`, die das gegebene Programm zur Lösung des Springerproblems ergänzt. Diese Bibliothek soll sowohl Datentypen (`struct board_t`, als auch Funktionen zur Modellierung eines Schachbretts mit n×n Feldern liefern, auf dem sich der Springer bewegen kann. Dabei soll für jede Position des Brettes festgehalten werden, in welchem Schritt der Springer das entsprechende Feld besucht hat. Die Größe `n` und die Startposition des Springers `startpos_x` und `startpos_y` werden zur Initialisierung übergeben. Die Felder des Brettes sollen dabei als dynamisches 2-dimensionales Feld der Form `int** fields` modelliert werden.

Die Funktion `void new_jump(struct board_t* b, int x, int y)` bewegt die Springer um

`x` Felder in der Horizontalen und um `y` Felder in der Vertikalen. Die neue Position des Springers ergibt sich automatisch aus der alten Position und der aktuellen Bewegung. Das neu besuchte Feld wird entsprechend markiert.

Mit `void remove_jump(struct board_t* b, int x, int y)` wird die Bewegung um (x, y) rückgängig gemacht und die alte Markierung entfernt.

Die Funktion `int isfree(struct board_t* b, int x, int y)` liefert zurück, ob der jeweilige Sprung von der aktuellen Position ausgehend erlaubt ist, d.h. noch nicht besucht worden ist. Beachten sie dabei, dass Sprünge über den Rand des Brettes nicht erlaubt sind.

`int visited_fields(struct board_t* b)` liefert die Anzahl der Felder, die bereits von dem Springer besucht wurden, zurück.

`void print_board(struct board_t* b)` gibt das aktuelle Schachbrett z.B. in folgender Form aus:

```
+---+---+---+---+---+
+  1+  4+ 11+ 16+ 25+
+---+---+---+---+---+
+ 12+ 17+  2+  5+ 10+
+---+---+---+---+---+
+  3+ 20+  7+ 24+ 15+
+---+---+---+---+---+
+ 18+ 13+ 22+  9+  6+
+---+---+---+---+---+
+ 21+  8+ 19+ 14+ 23+
+---+---+---+---+---+
```

> Hinweis: Mit dem Formatierungszeichen `"%3i"` kann die Länge der Ziffernausgabe von `printf` gesteuert werden.

Die Funktion `int init_board(struct board_t* b, int n, int x, int y);` soll eine Initialisierung durchführen, d.h. für die (global deklarierte) Datenstruktur des Schachbrettes (Übergabe per Zeiger als 1. Parameter) soll Speicherplatz für das dynamische 2- dimensionale Feld reserviert werden.

`void free_board(struct board_t *b)` soll den reservierten Speicher wieder freigeben.

Probieren Sie das vollständige Programm aus und überprüfen Sie einige der ausgegebenen Lösungen auf Korrektheit. Für $n = 5$ und Startposition $(0, 0)$ sollte Ihr Programm 304 verschiedene Lösungen finden.

Aufgabe 2 Fehlersuche in Programmen (2 Punkte)

In das folgende Programm haben sich einige syntaktische und semantische Fehler eingeschlichen, die Sie erkennen und korrigieren sollen.

> Hinweis: Für die Fehlersuche in komplexeren Programmen ist es sinnvoll, statt der Einführung von Hilfsausgaben einen Debugger zu verwenden. Dieser ermöglicht eine schrittweise Programmausführung, bei der in jedem Schritt eine Beobachtung aller Variablen im Sichtbarkeitsbereich durchgeführt werden kann. Ein prominenter Debugger für C-Programme ist dabei der *GNU Debugger* (`gdb`). Im Internet finden sich zahlreiche Einführungen in die Bedienung von `gdb`.

Finden und korrigieren Sie die im folgenden Programm vorhandenen Fehler. Protokollieren Sie die Fehler in einer „README“-Datei. Das Programm können Sie von unserer Homepage herunterladen außerdem ist es in die PDF-Datei eingebunden:

```c
#include <stdio.h>

int main(void) {
        int number1 /* first summand */
        int number2 /* second summand */

        printf("Bitte erste ganze Zahl eingeben: ");
        scanf("%i", number1);

        printf("Bitte zweite ganze Zahl eingeben: ");
        scanf("%i", number2);

        if (number1 = number2) {
                printf("Die eingegebenen Zahlen sind identisch.\n");
        }

        printf("Die Summe von %i und %i ist %i.\n",
                number1, number2, number1 + number2);

        return 0;
```

Aufgabe 3 Hello World in Assembler (6 Punkte [1 + 3 + 1 + 1])

> Hinweis: Die Inhalte dieser Aufgabe werden in der Vorlesung am Dienstag, den 24. Oktober 2023, fertig besprochen.

Für diese und die folgenden Assembler-Aufgaben verwenden Sie bitte den Netwide Assembler (NASM). Die ausführbare Datei heißt üblicherweise `nasm`. NASM benutzt als Standard-Einstellung die Intel-Syntax.

Denken Sie bitte auch daran, Ihre Programme stets gründlich und verständlich zu kommentieren.

a) (1 Punkt)

Gegeben ist das folgende, unkommentierte „Hello world"-Programm.

```
SECTION .data

str:        db `Hello world!\n`
strlen:        equ $ - str

num:        dq 1337

SECTION .text

global _start

_start:
        mov rax, 1
        mov rdi, 1
        mov rsi, str
        mov rdx, strlen
        syscall

        mov rax, 60
        mov rdi, 0
        syscall
```

Speichern Sie den Quellcode unter `helloworld-a.S` (auch zu finden auf der SysProg-Homepage oder eingebettet in diese PDF-Datei). Ergänzen Sie den Quellcode um aussagekräftige Kommentare. Assemblieren, linken und testen Sie dann das Programm. Die dazu notwendigen Schritte finden Sie in den Vorlesungsfolien.

b) (3 Punkte)

Kopieren Sie die Quellcodedatei aus Teilaufgabe a) nach `helloworld-b.S`. Ergänzen Sie das Programm so, dass der Wert der Variable `num` in einen String umgewandelt und in der Konsole ausgegeben wird.

> Hinweis: Sie erreichen dies durch eine wiederholte Division durch 10 unter Beachtung des Rests. Sie können zu diesem Zweck analog zu `str` einen String deklarieren, der als Puffer für die auszugebende Zahl dient.

Sie können davon ausgehen, dass es sich bei `num` um eine ganze, nicht negative Zahl handelt, also gilt `num` $\in \mathbb{N}_0$. Testen Sie ihr Programm mit verschiedenen Werten für `num`.

c) (1 Punkt)

Kopieren Sie ihre Lösung aus Teilaufgabe b) nach `helloworld-c.S`. Lagern Sie nun den Code zur Umwandlung einer Zahl des Typs `dq` in einen String in eine Funktion `printnumber` der Form

```
printnumber:
        ; ...
        ret
```

aus. Diese soll aus dem Hauptprogramm folgendermaßen aufgerufen werden:

```
mov rax, num
call printnumber
; Auszugebende Zahl wird in rax übergeben
; Funktion aufrufen
```

> Hinweis: Es gibt in Assembler keine echte Unterscheidung zwischen den Begriffen Funktion, Prozedur und Subprogramm. Ebenso gibt es keine vorgeschriebene Art, Parameter zu übergeben und Rückgabewerte zurückzugeben (deshalb gibt es Konventionen wie die *cdecl* für x86 32 Bit oder *AMD64 ABI* für x86 64 Bit).

d) (1 Punkt)

Beschreiben Sie, was die Befehle `call` und `ret` genau tun. Warum ist der Stack dafür wichtig?