Programmieren in C

Prof. Dr. Peter Thiemann
Hannes Saffrich
Sommersemester 2021

Universität Freiburg
Institut für Informatik

Übungsblatt 9

Abgabe: Montag, 28.06.2021, 9:00 Uhr morgens

In diesem Übungsblatt geht es um Enums, Unions und JSON.

Ihr Repository enthält für dieses Blatt wieder vorgegebene Dateien → `git pull --rebase`.

Aufgabe 9.1 (Unions und Enums; 3 Punkte)

Für diesen Aufgabenteil sind die Dateien `event.h`, `event.c` und `event_example.c` relevant.

In einer GUI library können verschiedene Events auftreten:

- die Maus wurde an einer bestimmten Position geklickt; oder
- eine Taste wurde auf der Tastatur gedrückt.

Definieren Sie in der `event.h` ein `struct Event`, dessen Werte entweder einen Mausklick oder einen Tastendruck repräsentieren. Verwenden Sie hierzu `union`, `enum` und `struct`.

Implementieren Sie die Funktionen

```
Event mouse_click_event(int x, int y);
Event key_press_event(char key);
```

die einen neuen Event aus der Mausposition bzw. der gedrückten Taste erzeugen.

Implementieren Sie die Funktion

```
void print_event(Event e);
```

die einen `Event` in Textform ausgibt. Die Ausgabe soll dabei für

```
print_event(mouse_click_event(2, 3));
print_event(key_press_event('x'));
```

wie folgt aussehen:

```
Mouse clicked at (2, 3).
Key 'x' pressed.
```

Übrigens: In dieser Aufgabe implementieren Sie genau was bei dem Rust-Code in Abbildung 1 im Hintergrund passiert.

```rust
// Those are not C enums, but full blown "Algebraic Data Types".
enum Event {
    MouseClick { x: i32, y: i32 },
    KeyPress { key: char },
}

fn print_event(e: Event) {
    // This is called "Pattern Matching".
    match e {
        Event::MouseClick { x, y } => println!("Mouse clicked at ({}, {}).", x, y),
        Event::KeyPress { key } => println!("Key '{}' pressed.", key),
    }
}

fn main() {
    print_event(Event::MouseClick { x: 2, y: 3 });
    print_event(Event::KeyPress { key: 'x' });
}
```

Abbildung 1: Rust-Version von Aufgabe 1

REST, HTTP, JSON und curl.

Kommunikation zwischen Servern im Web läuft häufig nach dem Prinzip des *Representational State Transfers (REST)*. Auf Wikipedia[1] wird dies wie folgt beschrieben:

> Any web service that obeys the REST constraints is informally described as RESTful. Such a web service must provide its Web resources in a textual representation and allow them to be read and modified with a stateless protocol and a predefined set of operations

Das bekannteste Beispiel ist hier die Kommunikation eines Browsers mit einem Webserver über das HTTP Protokoll. Der Client (Browser) kann Anfragen an den Server stellen und dieser antwortet dann dem Client. Der Server schließt die Verbindung dabei direkt wieder nach dem Beantworten der Anfragen (HTTP ist *stateless*).

Unter anderem unterstützt das HTTP-Protokoll[2] die folgenden beiden Arten von Anfragen, die auf Wikipedia wie folgt beschrieben werden:

- The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
- The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.

Gibt man im Browser die URI einer Webseite ein, so sendet dieser einen `GET`-Request an den Webserver, der mit einem `HTML`-Dokument antwortet.

Klick man im Browser auf einen Form-Button, z.B. beim Registrieren auf einer Webseite, so sendet dieser einen `POST`-Request an den Server mit dem gewünschten Benutzername und Passwort.

Wenn andere Programme mit Servern über HTTP kommunizieren, dann wird meistens über JSON statt HTML kommuniziert.

Da sich die verschiedenen URIs, die der Webserver unterstützt, grob wie Funktionsaufrufe verhalten, nennt man diese häufig *API Endpoints*.

[1] https://en.wikipedia.org/wiki/Representational_state_transfer
[2] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Highscore Server.

Für das Asteroiden-Spiel haben wir einen Webserver geschrieben, der die Highscores verwalten soll. Der Server ist unter der URI

`https://stream.inpro.informatik.uni-freiburg.de`

zu erreichen und unterstützt die folgenden API-Endpunkte:

- `POST /highscores` sendet ein einzelnes Spielergebnis als JSON zum Server, der das Spielergebnis in seine Datenbank einträgt.
- `GET /highscores` frägt den Server nach den Top 10 Highscores, der diese als JSON zurücksendet.
- `GET /highscores/<name>` frägt den Server nach den Top 10 Highscores für den Spieler `<name>`, der diese als JSON zurücksendet.

Für den API-Endpunkt `POST /highscores` erwartet der Server JSON der folgenden Form:

```
{ "name": "xyz", "points": 42, "distance": 230 }
```

Für die API-Endpunkte `GET /highscores` und `GET /highscores/<name>` antwortet der Server mit JSON der folgenden Form:

```
{
  "0": { "name": "xyz", "points": 42, "distance": 230 },
  "1": { "name": "abc", "points": 5, "distance": 50 },
  "2": { "name": "rst", "points": 0, "distance": 70 }
}
```

Das Kommandozeilenprogramm `curl` erlaubt es von Hand HTTP-Anfragen an einen Webserver zu senden und gibt dann die Antwort des Servers aus.

Installieren Sie sich `curl`, z.B. mit

```
$ sudo apt-get install curl
```

Der API-Endpunkt `GET /highscores` kann dann wie folgt aufgerufen werden:

```
$ curl -X GET https://stream.inpro.informatik.uni-freiburg.de/highscores
```


```
{"0":{"distance":960,"name":"XXX","points":65},"1":{"distance":1337,"name":"XXX",
"points":42},"2":{"distance":23,"name":"foo","points":42},"3":{"distance":660,
"name":"XXX","points":20},"4":{"distance":580,"name":"XXX","points":15},"5":
{"distance":1111,"name":"XXX","points":11}}
```


Das Programm aus dem nächsten Aufgabenteil nimmt an, dass Sie `curl` installiert haben.

Aufgabe 9.2 (Highscores; 11 Punkte)
Für diesen Aufgabenteil wurde ihr Repository wie folgt erweitert:

- Dateien: `game.c, game_lib.*`

  Die Musterlösung des Asteroiden-Spiels von Blatt 7, erweitert um einen Titel- und Gameover-Bildschirm. Die Änderungen beziehen sich effektiv nur auf die `game.c`.

- Dateien: `vec.*`

  Die Datenstruktur für dynamische Arrays von Blatt 5, die so abgeändert wurde, dass sie nicht mehr automatisch bei `vec_free` ihre Elemente freigibt, was problematisch für die JSON-Strukturen wäre, da es für diese nicht ausreicht `free` aufzurufen.

- Dateien: `game_highscores.*, game_highscores_test.c`

  Datenstrukturen für Highscores und Funktionen um JSON vom Server zu empfangen bzw an den Server zu senden. Diese sollen Sie in diesem Aufgabenteil erweitern.

- Dateien: `json_data.*, json_reader.*, json_printer.*, reader.*, memtools.*`

  Die JSON-Datenstrukturen aus der Vorlesung, ergänzt um einen fertig geschriebenen Parser und einen Printer, der die JSON-Strukturen wieder in Text umwandeln kann.

  Der Parser und Printer werden in der `game.c` bereits korrekt aufgerufen. Die Strukturen aus `json_data.*` müssen Sie aber für diese Aufgabe genau verstehen.

Ein Spielergebnis des Asteroiden-Spiels hat folgende Struktur:

```
typedef struct Highscore {
  char* name;    /* Player name; can be set on the title screen. */
  int points;
  int distance;
} Highscore;
```

Die Top 10 Highscores repräsentieren wir im Spiel durch einen `Vec*` dessen Elemente `Highscore`s sind.

Implementieren Sie in der `game_highscores.c` die Funktionen

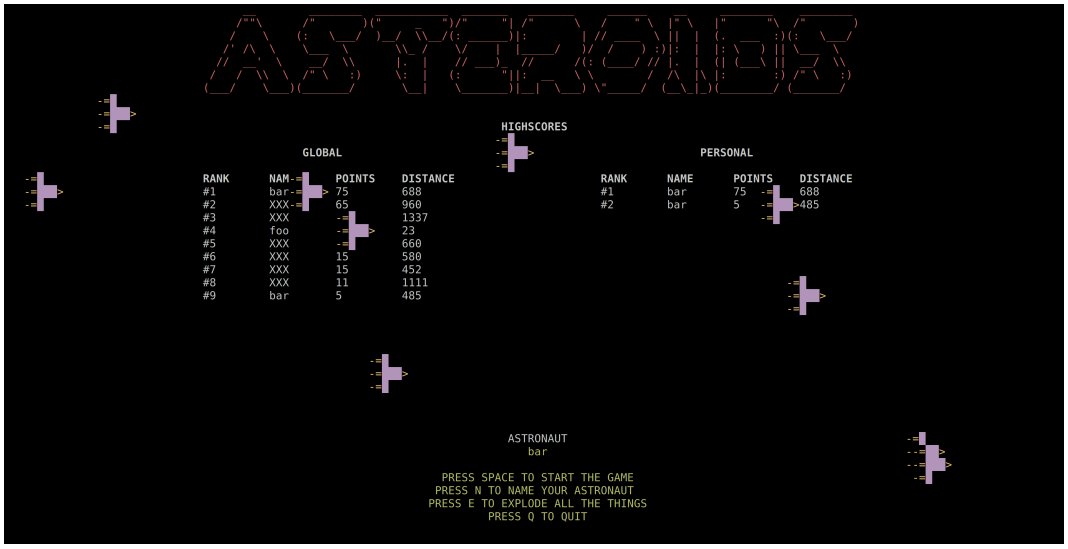
```
JsonValue* highscore_to_json(Highscore* highscore);  // Single highscore
Vec* json_to_highscores(JsonValue* highscores_json); // Many highscores
```

die zwischen den Datenrepräsentationen aus dem Spiel und des Servers konvertieren.

Testen Sie Ihre Funktionen in der `game_highscores_test.c` um den Server nicht mit sinnfreien Anfragen zu überfluten. Für `highscore_to_json` können Sie statt einem richtigen Unittest das erzeugte JSON einfach mit `json_fprint_value(stdout, 0, value)`

auf dem Terminal ausgeben. Die Funktion `json_fprint_value` wird in `json_printer.h` deklariert.

Sie können das Spiel bereits testen, obwohl es noch nicht vollständig ist. Nach Bearbeitung sieht der Titelscreen wie folgt aus:



Aufgabe 9.3 (Erfahrungen; 2 Punkte)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt in der Datei `erfahrungen.txt` (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Der Zeitaufwand *muss* dabei in der ersten Zeile und in exakt dem folgenden Format notiert werden, da wir sonst nicht automatisiert eine Statistik erheben können:

```
Zeitaufwand: 3:30

<...Andere Erfahrungen...>
```

Die Angabe `3:30` steht hier für 3 Stunden und 30 Minuten.