Programmieren in C

SS 2021

Vorlesung 10, Dienstag 29. Juni 2021

(Parsing, Enums, Unions, Funktionszeiger)

Prof. Dr. Peter Thiemann
Lehrstuhl für Programmiersprachen
Institut für Informatik
Universität Freiburg
Nach einer Formatvorlage von Prof. Dr. Hannah Bast

Die heutige Vorlesung

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit den Blättern — json und events
  - Punkte und Kriterien — Erinnerung + Klarstellung
- Inhalt
  - Recursive Descent Parsing — Rezept + Beispiel JSON1
  - Anwendung: vec_iteration — Funktionszeiger (#1)
  - **Ü10: Game of Life auf TUI**

Punkte und Erfahrungen 1/4

Punkte

Name	Minimum	Quartile 1	Median	Quartile 3	Maximum	Average	Mean Deriv	Variance	Samples
blatt01	0.00	16.00	16.00	16.00	16.00	14.07	3.02	21.26	269
blatt02	0.00	11.00	14.00	15.50	16.00	11.80	4.17	28.76	273
blatt03	0.00	5.50	14.00	15.50	16.00	10.52	5.39	37.50	274
blatt04	0.00	3.00	12.00	15.00	16.00	9.35	5.66	38.28	270
blatt05	0.00	0.00	11.50	15.00	16.00	8.88	6.21	44.00	273
blatt06	0.00	0.00	8.50	14.50	16.00	7.92	5.57	38.05	269
blatt07	0.00	0.00	16.00	22.00	22.00	12.68	8.58	87.63	272
blatt08	0.00	0.00	9.00	14.50	16.00	8.14	5.52	38.74	215

Zeitbedarf

Name	Minimum	Quartile 1	Median	Quartile 3	Maximum	Average	Mean Deriv	Variance	Samples
Blatt 01	0.33	2.0	3.75	5.0	25.25	4.03	1.89	8.69	235
Blatt 02	0.33	3.0	4.0	6.0	36.0	4.94	2.14	12.04	224
Blatt 03	0.0	4.5	6.0	8.0	24.0	6.44	2.39	10.82	217
Blatt 04	0.5	5.0	6.5	9.0	30.0	7.16	2.68	13.82	211
Blatt 05	0.5	5.5	7.5	9.0	29.0	7.79	2.61	14.79	194
Blatt 06	0.0	6.0	8.5	11.0	48.0	9.11	3.24	22.78	192
Blatt 07	0.0	5.0	6.5	9.5	48.0	7.72	3.02	23.24	184
Blatt 08	0.0	4.0	6.0	8.0	25.0	6.28	2.75	13.76	180
Blatt 09	0.0	4.0	6.78	9.0	36.0	7.19	3.27	22.04	157

Punkte und Erfahrungen 2/4

UNI FREIBURG

```
Files matching regexes:

  40   schwer|schwierig|anspruch|aufwendig|fordernd|hart|viel zeit|lange gedauert
  2    nicht schwer|nicht schwierig|nicht anspruch|nicht aufwendig|unaufwendig|nich
  49   cool|nett|spaß|gut|schön|toll|super
  5    nicht cool|uncool|nicht nett|keinen spaß|nicht gut|nicht schön|unschön|nicht
  8    unklar|verwirrend

_____________________________________________________________________________
```

Punkte und Erfahrungen 3/4

UNI FREIBURG

Zeitaufwand: 10:00

Sehr anstrengedes Blatt. Im Endeffekt ware die Aufgaben gar nicht mal soo tragisch aber, bis man sich mal mit der Struktur der Json Objects und Struct und
und und da hat man erst mal gebraucht. Ich würde die Struktur vom Blatt auch gerne etwas in Frage stellen, habe aber auch kein Problem da falsch zu liegen.
Doch nach meiner Sicht sind die Objects einfach so extrem verschachtelt das muss doch nic ht sein oder ? Andererseits vielleicht ja schon, jedoch ist der
Zugriff in C einfach nicht so schön in Python kann man einfach json_obj.highscores.name o der so machen das ist dann halt sehr angenehm in C wohl nicht so.

Punkte und Erfahrungen 4/4

UNI FREIBURG

Zeitaufwand: 4:00

War ein angenehmes Blatt, ich finde cool wie sich die game.c durch die ganze Studienleistung zieht und immer besser wird. Die Aufgabenstellung war diesmal freier und die Richtung nicht so klar wie sonst, die meiste Verwirrung war, sich im gegebenen Code anfangs zurecht zu finden. Ich habe zB. viele json_data Funktionen übersehen, die mir das Leben von Anfang an leichter gemacht hätten.

Ich hätte nicht erwartet, dass die Highscore Liste online sich wirklich ändert, die Vorbereitung der Blätter ist wie immer krass. Sieht man sich die Highscores an sollte das nächste Übungsthema Authentifizierung, Fuzzing und Sicherheit sein ;)

Prüfungstechnisches 1/1

UNI FREIBURG

- Wiederholung aus Vorlesung #1
  - Sie bekommen wunderschöne Punkte, maximal 16 pro Übungsblatt, das sind maximal 176 Punkte für Ü1 – Ü11
  - Für das Projekt gibt es maximal 80 Punkte
  - Macht insgesamt 256 Punkte
  - **Außerdem:** Zum Bestehen müssen mindestens 88 Punkte aus den Übungsblättern (Ü1 – Ü11) und mindestens 40 Punkte im Projekt erreicht werden
  - **Außerdem 2:** Sie müssen sich mindestens einmal mit Ihrem Tutor / Ihrer Tutorin treffen (scheint zu laufen)

Parsing 1/8

UNI FREIBURG

- Erinnerung: JSON
  - Mensch- und Maschinenlesbares Datenaustauschformat
  - Einfaches Beispiel

```
{ "fruit": "Apple", "size": "Large", "color": "Red" }
```

  - Geschachteltes Beispiel

```
{ "host": "localhost",
  "port": 3030,
  "public": "../public/",
  "paginate": { "default": 10, "max": 50 },
  "mongodb": "mongodb://localhost:27017/api"
}
```

Parsing 2/8

- Wiederholung: Definition JSON0 (BNF Format)

```
Object ::=
    '{' '}'
    '{' Members '}'
Members ::=
    Member
    Member ',' Members
Member ::=
    String ':' Value
Value ::=
    Number
    String
    Object
```

Object, Members, Member, String, Value, Number sind Variable

'{' '}' ',' ':' sind Terminale: Zeichen, die so in der Eingabe vorkommen müssen

Übereinanderstehende Zeilen sind Alternativen

Parsing 3/8

UNI FREIBURG

- BNF (Backus-Naur Form; Backus-Normal Form)
  - Benannt nach John Backus (Turing Award 1977) und Peter Naur (Turing Award 2005) zur Beschreibung der Syntax von ALGOL60, einem Vorgänger von C
  - Fast jede Sprache hat heutzutage eine Beschreibung in BNF
  - Variable werden definiert durch Regeln der Form

```
Object ::=
    '{' '}'
    '{' Members '}'
```

  - Ein Vorkommen einer Variable wird durch eine der rechten Regelseiten ersetzt.
  - Diese Ersetzung geht solange weiter bis nur noch Symbole ohne Regeln vorhanden sind.

Parsing 4/8

UNI FREIBURG

- Beispiel

Object

-> ‘{‘ Members ‘}‘

-> ‘{‘ Member ‘}‘

-> ‘{‘ String ‘:‘ Value ‘}‘

-> ‘{‘ String ‘:‘ Number ‘}‘

  - Die jeweils ersetzte Variable ist in rot angezeigt
  - Zum Schluss nur noch Terminalsymbole und primitive Variablen ohne Regeln
  - Die muss ein Parser erkennen und einlesen

Parsing 5/8

UNI FREIBURG

- Lesen der primitiven Variablen -- String

  JsonValue *parse_string(reader * input)

  – Vorbedingung: Eingabe nicht beendet
  – Prüfe das nächste Zeichen auf “
  – Misserfolg, falls es ein anderes ist
  – Sonst lies Zeichen bis “ oder Dateiende

Parsing 6/8

UNI FREIBURG

- Lesen der primitiven Variablen -- Number

  JsonValue *parse_number(reader *input)

  – Vorbedingung: input zeigt auf – oder Ziffer

  – Lese Ziffern und andere Bestandteile von Zahlen mit Bibliotheksfunktion

Parsing 7/8

UNI FREIBURG

- Funktionsweise der Primitiven
  - Alle nach dem gleichen Muster
  - Test in parse_value stellt sicher, dass das erste Zeichen zum Primitiv passt
  - Liefert eindeutiges Ergebnis, weil jede Alternative mit einer anderen Art Zeichen beginnt -> Design des JSON Formats!

Parsing 8/8

UNI FREIBURG

- Generelle Vorgehensweise
  - Jede Variable entspricht einer (rekursiven) Funktion
  - Alle Funktionen müssen vorab deklariert werden, weil sie sich wechselseitig aufrufen
  - Auf der rechten Regelseite
    - Jedes Vorkommen einer Variable = Aufruf der Funktion
    - Jedes Vorkommen eines Terminalsymbols = Aufruf von expect () mit dem entsprechenden Symbol
  - Rückgabewerte müssen auf Misserfolg getestet werden
  - Bei Alternativen beginne mit der ersten und springe bei Misserfolg am Anfang jeweils zum nächsten

Bäume 1/6

UNI FREIBURG

- Am Beispiel von JSON0
  - Ein JSON0 Objekt enthält Members, nämlich einen Vektor von Member
  - Ein JSON0 Member besteht aus
    - Einem Attributnamen (String) und
    - dem zugehörigen Wert (Value)
  - Ein JSON0 Wert ist entweder
    - Ein String
    - Eine Zahl (Number) oder
    - ein Objekt (Object)

Bäume 2/6

UNI FREIBURG

- Member wird durch ein **struct** dargestellt
  - Ein JSON0 Member besteht aus
    - Einem Attributnamen (String) und
    - dem zugehörigen Wert (Value)

```
struct JsonMember {
  char* name;
  JsonValue* value;
};
```

  - Ein „Name-Wert Paar" (name value pair)

Member_name	Member_value

Bäume 3/6

UNI FREIBURG

- Ein Objekt wird durch ein **struct** dargestellt
  - Ein JSON0 Objekt enthält einen Vektor von Members

```
struct JsonObject {
  Vec* members;
};
```

Bäume 4/6

UNI FREIBURG

- Für einen Wert gibt es mehrere Alternativen
  - Ein JSON0 Wert ist entweder
    - Ein String
    - Eine Zahl (Number) oder
    - ein Objekt (Object)
  - Wir müssen uns merken, welche Art von Wert vorliegt. In C mit einem **enum** (Enumeration, Aufzählung) Typ.

```
typedef enum JsonValueType {
  JSON_NUMBER,
  JSON_STRING,
  JSON_OBJECT
} JsonValueType;
```

Bäume 5/6

UNI FREIBURG

- Abspeichern der Alternativen
  - Ein JSON0 Wert ist entweder
    - Ein String
    - Eine Zahl (Number) oder
    - ein Objekt (Object)
  - Wir brauchen einen Speicherbereich, der groß genug ist, dass ein String oder eine Zahl oder ein Objekt(zeiger) abgelegt werden kann.
  - Das geht mit einem **union** Typ.

Bäume 6/6

UNI FREIBURG

- Abspeichern der Alternativen
  - Ein JSON0 Wert ist ein String, eine Zahl oder ein Objekt

```
struct JsonValue {
  JsonValueType type;
  union {
    double as_number;
    char* as_string;
    JsonObject* as_object;
  } value;
};
```

  - Zugriff auf die Alternativen der union erfolgt wie bei struct mit `my_value.value.as_number`
  - Aber nur nach test `my_value.value == JSON_NUMBER!`

Funktionszeiger 1/4

UNI FREIBURG

- Erinnerung: int Vektor
  - Datenstruktur aus Blatt 5
  - Dynamisches Array zur Aufnahme von int Werten
  - Später erweitert auf beliebige (Zeiger-) Werte
  - Jetzt Ivec zur Unterscheidung
- Aufgabe: Wende eine Operation auf alle Elemente eines int Vektors an
  - Drucke alle Elemente aus
  - Erhöhe jedes Element um 1
  - Ersetze jedes Element durch sein Vorzeichen
  - Usw.

Funktionszeiger 2/4

UNI FREIBURG

- Aufgabe
  - Gegeben ein Vektor und eine Operation
  - Wende diese Operation auf alle Elemente des Vektors an
- Ansätze
  1. Schreibe für jede Operation eine Schleife an der Stelle, wo sie benötigt wird
  2. Schreibe für jede Operation eine Funktion mit dieser Schleife
  3. Schreibe eine Funktion mit einer Schleife, die für alle Operationen verwendet wird
- Dafür brauchen wir Funktionszeiger!

Funktionszeiger 3/4

UNI FREIBURG

- Signatur

```
/* Apply function `func` to every element of `xs` */
void ivec_map(Ivec *xs, int (*func) (int));
```

  - Das Argument func ist ein Zeiger…
  - Auf eine Funktion, die int zurückliefert, und
  - Ein int als Argument nimmt!

- Beispielhafter Aufruf
  - In stdlib.h gibt es eine passende Funktion

```
int abs(int j);
```

  - Aufruf mit

```
ivec_map(xs, &abs); // oder einfach abs tut auch
```

Funktionszeiger 4/4

- Alternative

```
/* Apply function `func` to every element of `xs` */
void ivec_iter(Ivec *xs, void (*func) (int*));
```

  - Das Argument func ist ein Zeiger…
  - Auf eine Funktion, die void zurückliefert, und
  - Ein int als Argument nimmt!

- Beispielhafter Aufruf
  - In stdlib.h gibt es eine passende Funktion

```
void print_me(int *ip) { printf("%d\n", *ip); }
void do_abs(int *ip) { *ip = abs(*ip); }
```

  - Aufruf mit

```
ivec_iter(xs, &do_abs);
```

Literatur / Links

- Enums

https://www.geeksforgeeks.org/enumeration-enum-c/

- Unions

https://www.geeksforgeeks.org/union-c/

- JSON

http://json.org/