Programmieren in C

SS 2021

Vorlesung 9, Dienstag 22. Juni 2021

(Parsing, enums, unions)

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

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem Ü8 — Dateien
- Inhalt
  - Recursive Descent Parsing — Prinzip + Beispiel JSON0
  - Modellierung von JSON0 — C: enums und unions
  - **Ü9: Highscoreverwaltung**

Erfahrungen mit dem Ü8 1/3

UNI FREIBURG

- 182 Erfahrungen, 180 mit lesbarer Zeit
- Zeitstatistik (in Stunden)

Min	Q1	Med	Q3	Max	Avg	MD	Var
0.0	4.0	6.0	8.0	25.0	6.28	2.75	13.76

-

#	Schlüsselworte
40	schwer|schwierig|anspruch|aufwendig|fordernd|hart|viel zeit|lange gedauert
1	nicht schwer|nicht schwierig|nicht anspruch|nicht aufwendig|unaufwendig|nicht fordernd|nicht hart|nicht viel zeit|nicht lange gedauert
70	cool|nett|spaß|gut|schön|toll|super
4	nicht cool|uncool|nicht nett|keinen spaß|nicht gut|nicht schön|unschön|nicht toll|nicht super
4	unklar|verwirrend

Erfahrungen mit dem Ü8 2/3

UNI FREIBURG

- Zusammenfassung / Auszüge
  - (Zeitaufwand: 18:00) Das Blatt war nicht einfach.

    Etwas genauer?
  - (Zeitaufwand: 6:00) Wer dachte, es sei eine gute Idee, dass ein Linebreak in Linux \n aber in windows \r\n und auf mac \r ist.

    😬
  - (Zeitaufwand: 6:30) Sehr interessantes blatt. Datenstruktur mit doppelt verlinkter Liste gibt richtig viel her.

    Nächstesmal gern mit Sentinel? ;)

    Leider den 2. teil wegen Corona-Impfkater nichtmehr

    hinbekommen. Aber cooles blatt :)

    I feel for you! Ich bekomme #2 am 6.7. um 10:07 …

Erfahrungen mit dem Ü8 3/3

UNI FREIBURG

- Zusammenfassung / Auszüge
  - (Zeitaufwand: 11:00) Ich hatte bei der ersten Aufgabe erstmal einige Schwierigkeiten mit dem list_free, bis ich meinen Fehler gefunden hatte (ich habe vergessen die Länge der Liste zu reduzieren....)

    Ansonsten war die erste Aufgabe durch das in VL vorgestellte gut machbar.

    Hat sich gelohnt.
  - (Zeitaufwand: 6:40) Was ist mit den "Wenn malloc fehlschlaegt, koennen Sie das Programm eingach mit exit(1) abbrechen"? habe ich nicht benutzt. sollte ich?

    Ja, das vereinfacht das Programm, weil nicht immer auf NULL getestet werden muss. In der neuen Ü gibt es eine Vorgabe malloc_exit(), die das schon macht!

Parsing 1/8

UNI FREIBURG

- Motivation: JSON (siehe https://json.org/)
  - JSON (JavaScript Object Notation) ist ein textuelles Datenaustauschformat.
  - Es ist für Mensch und Maschine einfach zu lesen und zu schreiben.
  - JSON repräsentiert Objekte, Felder und primitive Daten
  - Wichtig: beliebig geschachtelt!
  - Wir wollen eine Teilmenge von JSON einlesen **(parsen)**.
    - Ohne Arrays, null, true, false
    - Zahlen und Strings vereinfacht

Parsing 2/8

UNI FREIBURG

- Das JSON0 Format
  - 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 3/8

UNI FREIBURG

- Das JSON0 Format
  - Vergleichbar mit struct
  - Jedes struct kann in JSON repräsentiert werden

```
struct Fruit {
  char *fruit; char *size; char *color;
}
```

→

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

  - Unterschied: die Feldnamen sind in JSON nicht festgelegt
  - Zur Repräsentation eines beliebigen JSON Werts reicht ein Strukt nicht aus!

Parsing 4/8

UNI FREIBURG

- Definition JSON0 (BNF, kontextfreie Grammatik)

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

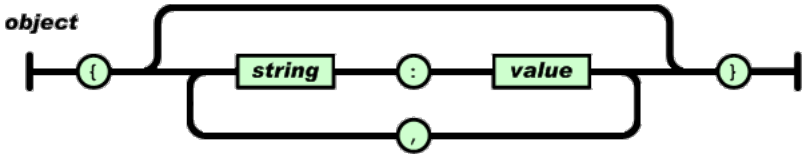
Object, Members, Member, String, Value, Number sind **Variable**

'{' '}' ',' ':' sind **Terminale**, d.h. Zeichen, die in der Eingabe genauso vorkommen müssen

Übereinanderstehende Zeilen sind Alternativen

Parsing 5/8

- Graphische Darstellung der Regeln für Object



  - Railroad Diagram
  - Jeder Durchlauf von links nach rechts druckt ein gültiges Objekt
  - Dabei werden Zeichen ausgegeben und Variable wie Funktionen „aufgerufen“
  - Einlesen (parsen) geht genauso, nur dass die Zeichen gelesen und verglichen werden
  - Richtschnur für Implementierung: eine (ggf. rekursive) Funktion pro Variable

Parsing 6/8

UNI FREIBURG

- Einlesen
  - Vereinfachung: lese vollständige Datei in einen String
  - Verarbeite Eingabe über

```
struct reader {
  const char *p;
} *input;
```

  - Prüfen von Zeichen über `*input->p`

Parsing 7/7

UNI FREIBURG

- Überlesung von Leerzeichen etc
  - #include <ctype.h>
  - Enthält Klassifikationsfunktionen für char
    - isspace(), isalpha(), isdigit(), …
  - Überlesen durch eine while-Schleife:

```
void skipws(reader *input) {
  while (isspace (peek(input))) {
    next (input);
} }
```

  - Anschauen eines Zeichens

```
#define peek(input) (*(input)->p))
```

  - Jeder Aufruf dieses Makros erzeugt eine Kopie des Codes!

JSON Repräsentation 1/4

UNI FREIBURG

- Definition JSON0 (BNF, kontextfreie Grammatik)

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

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

'{' '}' ',' ':' sind **Terminale**, d.h. Zeichen, die in der Eingabe genauso vorkommen müssen

Übereinanderstehende Zeilen sind Alternativen

JSON Repräsentation 2/4

UNI FREIBURG

- Ein JSON Objekt ist ein Vektor von Members

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

- Ein Member besteht aus einem String und einem Value

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

JSON Repräsentation 3/4

UNI FREIBURG

- Ein JSON Value ist ein String, eine Zahl oder ein Objekt

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

- C Konzept dazu
  - Aufzählungstyp (enum)
  - Definiert eine Liste von Konstanten
  - Unterschiedliche Zahlen

JSON Repräsentation 4/4

UNI FREIBURG

- Ein JSON Value ist ein String, eine Zahl oder ein Objekt

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

- C Konzept: Vereinigungstyp (union)
  - Syntax wie struct, aber die „Felder" sind Alternativen
  - Speicherbereiche der "Felder" überlappen!

Literatur / Links

- Alles zu stdio

https://www.tutorialspoint.com/c_standard_library/stdio_h.htm

- Alles zu ctype

https://www.tutorialspoint.com/c_standard_library/ctype_h.htm

- Alles zu union

https://en.cppreference.com/w/c/language/union