Übungsstunde

Woche 09

Adel Gavranović
adel.gavranovic@inf.ethz.ch

Overview

Heutige Themen

Intro

Rekursion

(Extended-)Backus-Naur-Form

Structs

Feedback

Outro

Links

- polybox zum Material für die Übungsstunden
- Mail an Assistenten

Kommentare zu [code]expert

Allgemein

Kommentare zu [code]expert

Allgemein

- Wenn etwas auf den Vorlesungsfolien war, dürft ihr es verwenden (ausser es wird explizit verboten)
- Vergesst `std::endl` am Ende von Output nicht
- keine Kommentare $\Longrightarrow$ weniger Punkte
- betrügt bei den Aufgaben nicht, ihr schadet euch *wirklich* nur selbst
- es gab einen Testinput bei der Aufgabe E7:T2 (”-1 -1”) der faulty war und das war unser Fehler. Entschuldigung

Kommentare zu [code]expert

E7:T1: ”Const and reference types”

Kommentare zu [code]expert

E7:T1: ”Const and reference types”

- `print(...)` $\Longrightarrow$ `print(const ...)`
- gibt es noch Fragen bezüglich dieser Aufgabe (nach Abgleichen mit der ML)?
- konnte keine Aufgabe dazu in den bisherigen Prüfungen finden, also no worries ;)
- Kurzeinführung zu {Function scopes} ”Zombie Variables”

Kommentare zu [code]expert

E7:T1: ”Const and reference types”

- `print(...)` $\Longrightarrow$ `print(const ...)`
- gibt es noch Fragen bezüglich dieser Aufgabe (nach Abgleichen mit der ML)?
- konnte keine Aufgabe dazu in den bisherigen Prüfungen finden, also no worries ;)
- Kurzeinführung zu {Function scopes} ”Zombie Variables”

E7:T2: ”Number of Occurrences”

Kommentare zu [code]expert

E7:T1: "Const and reference types"

- `print(...)` $\Longrightarrow$ `print(const ...)`
- gibt es noch Fragen bezüglich dieser Aufgabe (nach Abgleichen mit der ML)?
- konnte keine Aufgabe dazu in den bisherigen Prüfungen finden, also no worries ;)
- Kurzeinführung zu {Function scopes} "Zombie Variables"

E7:T2: "Number of Occurrences"

- ihr könnt `return` s von Funktionen direkt in das `std::cout` packen, das spart oft eine Initialization und spart Zeilen
- viele haben den Nutzen von Referenzen nicht ganz verstanden: `Sequence = read_seq(vec& sequence)` mit `vec& read_seq(vec& sequence)` ergibt wenig Sinn, lieber `void read_seq(vec& sequence)`
- wenn eine Funktion die Inputparameter (Vektoren, Matrizen, Objects, etc.) nicht verändern soll, dann benutzt `const`

Kommentare zu [code]expert

E7:T3: ”Longest Increasing Subsequence”

Kommentare zu [code]expert

E7:T3: ”Longest Increasing Subsequence”

- sehr gute Arbeit, insb. bei den Kommentare. Bin stolz auf euch!
- nutzt `const`
- ich war bei den meisten recht ausführlich beim Feedback bei der Aufgabe T2, aber bei T3 eher weniger. Wenn ihr ausführlicheres Feedback zu T3 wollt, so bitte schreibt mir eine Mail und ich schau es mir sehr gerne noch einmal detailliert an

Kommentare zu [code]expert

E7:T3: ”Longest Increasing Subsequence”

- sehr gute Arbeit, insb. bei den Kommentare. Bin stolz auf euch!
- nutzt `const`
- ich war bei den meisten recht ausführlich beim Feedback bei der Aufgabe T2, aber bei T3 eher weniger. Wenn ihr ausführlicheres Feedback zu T3 wollt, so bitte schreibt mir eine Mail und ich schau es mir sehr gerne noch einmal detailliert an

E7:T4: ”Sorting by Swapping”

Kommentare zu [code]expert

E7:T3: ”Longest Increasing Subsequence”

- sehr gute Arbeit, insb. bei den Kommentare. Bin stolz auf euch!
- nutzt `const`
- ich war bei den meisten recht ausführlich beim Feedback bei der Aufgabe T2, aber bei T3 eher weniger. Wenn ihr ausführlicheres Feedback zu T3 wollt, so bitte schreibt mir eine Mail und ich schau es mir sehr gerne noch einmal detailliert an

E7:T4: ”Sorting by Swapping”

- schaut euch die ML an um eine elegantere implementation zu sehen, insb. wie man überprüft, ob die Sequenz nun sortiert ist

Fragen zu [code]expert eurerseits?

Lernziele

- ☐ bestimmen können, ob eine Zeichenkette gegebene EBNF-Regeln erfüllt
- ☐ EBNF-Regeln schreiben können, die nur gegebene Zeichenketten akzeptiert
- ☐ `struct` verstehen und implementieren können

Call Graphs

Eine Art Funktionsaufrufe zu `foo()` zu visualisieren

Call Graph for `power(x, 7)`

```
1  // PRE:  base x, power n
2  // POST: n'th power of x
3  unsigned int power(const int x, const unsigned int n){
4      if(n == 0){
5          return 1;
6      }else if(n == 1){
7          return x;
8      }
9
10     return x*power(x, n-1);
11 }
12 ...
13     std::cout << power(x, 7) << std::endl;
14 ...
15 // How will the "Call Graph" look like for this function?
16 // How many times will power() be called in total?
```

You've got the `power(x, n)`

Aufgabe

- Come up with a better (fewer recursive function calls) implementation of the `power()` -function with pen and paper
  Hint: $x^n = x^{\frac{n}{2}} \cdot x^{\frac{n}{2}}$
- Implement it in `[code]`expert in groups

You've got the `power(x, n)`

Aufgabe

- Come up with a better (fewer recursive function calls) implementation of the `power()` -function with pen and paper
  Hint: $x^n = x^{\frac{n}{2}} \cdot x^{\frac{n}{2}}$
- Implement it in `[code]`expert in groups
- Share your results and analyze the number of function calls your solution does. Argue over which implementation is better

Musterlösung zu ”Power Function”

```
1  // PRE:  n >= 0
2  // POST: result == x^n
3  unsigned int power (const int x, const unsigned int n){
4      if (n == 0) {                  // x^0 = 0
5          return 1;
6      } else if (n == 1) {           // x^1 = 1
7          return x;
8      } else if (n % 2 == 0) {       // x^n = x^(n/2) * x^(n/2)
9          int tmp = power(x, n/2);// recursive part
10         return tmp*tmp;
11     } else {                       // x^n = x^(n-1) * x
12         return x*power(x, n-1); // for non-power-of-two
13     }
14 }
```

Die Funktion wird sich selbst maximal $2\log_2(n+1) - 1$ mal aufrufen, d.h. nur *logarithmisch* oft im Vergleich zu *linear* oft bei der vorherigen implementation.

Fragen/Unklarheiten?

Das Konzept ”(E)BNF”

- eine Art, Input zu ”parsen” (mit `std::cin`, oder via Dateien) und zu schauen, ob sie *valide nach einem (E)BNF* sind

Das Konzept ”(E)BNF”

- eine Art, Input zu ”parsen” (mit `std::cin` , oder via Dateien) und zu schauen, ob sie *valide nach einem (E)BNF* sind
- funktioniert *rekursiv*

Das Konzept ”(E)BNF”

- eine Art, Input zu ”parsen” (mit `std::cin` , oder via Dateien) und zu schauen, ob sie *valide nach einem (E)BNF* sind
- funktioniert *rekursiv*
- das **E** in EBNF steht für *extended*. **E**BNFs bieten zusätzliche (kürzere, bessere) Arten, einen *validen Input* zu beschreiben

Das Konzept ”(E)BNF”

- eine Art, Input zu ”parsen” (mit `std::cin` , oder via Dateien) und zu schauen, ob sie *valide nach einem (E)BNF* sind
- funktioniert *rekursiv*
- das **E** in EBNF steht für *extended*. **E**BNFs bieten zusätzliche (kürzere, bessere) Arten, einen *validen Input* zu beschreiben
- viele Aufgaben werden lauten *”is xoo-xooxoo valid according to the given EBNF?”*

Das Konzept ”(E)BNF”

- eine Art, Input zu ”parsen” (mit `std::cin`, oder via Dateien) und zu schauen, ob sie *valide nach einem (E)BNF* sind
- funktioniert *rekursiv*
- das **E** in EBNF steht für *extended*. **E**BNFs bieten zusätzliche (kürzere, bessere) Arten, einen *validen Input* zu beschreiben
- viele Aufgaben werden lauten *”is xoo-xooxoo valid according to the given EBNF?”*
- sind also quasi *Regeln für erlaubte Sätze* (welche sehr komisch aussehen...)

Das Konzept ”(E)BNF”

- eine Art, Input zu ”parsen” (mit `std::cin`, oder via Dateien) und zu schauen, ob sie *valide nach einem (E)BNF* sind
- funktioniert *rekursiv*
- das **E** in EBNF steht für *extended*. **E**BNFs bieten zusätzliche (kürzere, bessere) Arten, einen *validen Input* zu beschreiben
- viele Aufgaben werden lauten *”is xoo-xooxoo valid according to the given EBNF?”*
- sind also quasi *Regeln für erlaubte Sätze* (welche sehr komisch aussehen...)
- Videoempfehlungen: ▸ EBNF I ▸ EBNF II
  (kleiner Fehler in EBNF I @6:00: in `digit` sollten alle Ziffern durch | getrennt werden

BNF Beispiel ”Aa_”

Wir möchen ein BNF definieren, welche folgende Regeln umfasst:

Rules

Alphabet = {’A’, ’a’, ’_’}
’A’ can only appear directly after an underscore or as the very first symbol. And underscores cannot occur in pairs and cannot be placed as the very first or the very last symbol.

So ist beispielsweise ”Aaaaa_aa” valid, aber nicht ”AaaAa”. Unsere Aufgabe ist es, ein BNF zu entwickeln, welches all diese Anforderungen beihaltet.

BNF Example ”Aa_”

BNF

```
seq        = term | term '_' seq
term       = 'A' | 'A' lowerterm | lowerterm
lowerterm  = 'a' | 'a' lowerterm
```

Beim Überpüfen, ob ein *Satz* valid ist, versucht es Stück für Stück zu dekonstruieren (mit dem gegebenen EBNF). Dieses BNF hat 3 Regeln:
Die erste und letzte haben jeweils zwei Alternativen, die zweite sogar drei. Dieses BNF hat 3 *non-terminale* Symbole (`seq`, `term`, `lowerterm`) und 3 *terminale* Symbole (`'A'`, `'a'`, `'_'`)

Fragen/Unklarheiten?

EBNF

Aufgabe

Schreibt das vorherige BNF von der vorherigen Folio ein ein EBNF um, mithilfe der folgenden Syntax:

- `{...}`: hier darf der Inhalt $n \in \{\mathbb{N}_0\}$ mal wiederholt werden
- `[...]`: hier darf der Inhalt $b \in \{0, 1\}$ mal wiederholt werden

EBNF

Aufgabe

Schreibt das vorherige BNF von der vorherigen Folio ein ein EBNF um, mithilfe der folgenden Syntax:

- `{...}`: hier darf der Inhalt $n \in \{\mathbb{N}_0\}$ mal wiederholt werden
- `[...]`: hier darf der Inhalt $b \in \{0, 1\}$ mal wiederholt werden

EBNF

```
seq   = term ['_' seq]
term  = 'A' {'a'} | 'a' {'a'}
```

Exercise ”Valid Words”

Aufgabe

```
seq   = term ['_' seq]
term  = 'A' {'a'} | 'a' {'a'}
```

Which of the following concatenations are *valid* `seq`s in the sense of the EBNF above?

1. `A`
2. `a`
3. `_`
4. `Aaa`
5. `aaA`
6. `A_A`
7. `Aa_Aa`

Exercise ”Valid Words”

Aufgabe

```
seq   = term ['_' seq]
term  = 'A' {'a'} | 'a' {'a'}
```

Which of the following concatenations are *valid* `seq`s in the sense of the EBNF above?

1. `A`
2. `a`
3. `_`
4. `Aaa`
5. `aaA`
6. `A_A`
7. `Aa_Aa`

(valid: 1, 2, 4, 6, 7)

Exercise ”Valid Words” Helper Functions

```
1  // PRE:  valid input stream input
2  // POST: returns true if further input is available
3  //       otherwise false
4  bool input_available(std::istream& input);
5
6  // PRE:  valid input stream input
7  // POST: the next character at the stream is
8  //       returned (but not consumed) if no input is
9  //       available, 0 is returned
10 char peek(std::istream& input);
```

Exercise ”Valid Words” Helper Functions

```
1  // POST: leading whitespace characters are extracted
2  //       from input, and the first non-whitespace character
3  //       is returned (but not consumed) if an error or end
4  //       of stream occurs, 0 is returned
5  char lookahead(std::istream& input);
6
7  // PRE:  Valid input stream input, expected > 0
8  // POST: If ch matches the next lookahead then it is consumed and
9  //       true is returned otherwise no character is consumed and
10 //       false is returned
11 bool consume(std::istream& input, char expected);
```

Code Example ”Valid Words”

Aufgabe

Versucht euch am Code Example ”Valid Words”

Fragen/Unklarheiten?

Structs

Ein `struct` ist ein Bündel von Zeugs

Structs

Ein `struct` ist ein Bündel von Zeugs

- das können Variablen, Funktionen, weitere structs und viel mehr sein (”Members”)
- Typen müssen nicht dieselben sein
- bieten uns eine Möglichkeit neue ”Objekte” zu definieren, z.B. einen eigenen Zahlentypen oder mathematische Objekte wie Geraden, Quadrate, Kreise, etc.

Struktur von `struct`

```
1  // Definition of the new object
2  struct Person {
3      unsigned int age;
4      std::string field;
5      std::vector<int> lucky_nums;
6  };
7
8  int main () {
9      Person Adel   = {24, "Computational Science", {1,8,4,8}};
10     Person Clone  = Adel;
11     Person Jules  = {23, "Linguistics", {1,3,1,2}};
12     std::cout << "Adel's " << Adel.age <<
13                  " years old\n" << std::endl;
14
15     return 0;
16 }
```

Fragen/Unklarheiten?

Aufgabe ”Geometry Exercise”

Aufgabe

- öffnet die Aufgabe ”Geometry Exercise” auf [code]expert
- versucht sie zu lösen
- helft einander

Euer Feedback an mich

▸ Link zur Umfrage

Tipps für [code]expert

- Orientiert euch an den Code Examples von letzter Woche

Allgemeine Fragen?

Bis zum nächsten Mal

- ☐ macht eure Hausaufgaben
- ☐ bleibt gesund