Informatik I - Übung 5

Pascal Schärli

pascscha@student.ethz.ch

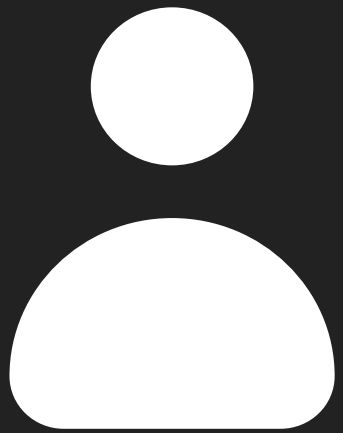
22.03.2019

ETH zürich

Was gibts heute?

- Self-Assesment
- Nachbesprechung
- Best-Of Vorlesung:
  - Pre/Post Conditions
  - Funktionen
  - Stepwise Refinment (&Program of the Week)
- Vorbesprechung

Self-Assesment



Nachbesprechung

Pascal Schärli 22.03.2019

Bild: look again by corpus delicti from the Noun Project

ETH zürich

Loop Analysis

```
unsigned int n;
std::cin >> n;

unsigned int x = 1;
if (n > 0) {
  unsigned int k = 0;
  bool e = true;
  do {
    if (++k == n) {
      e = false;
    }
    x *= 2;
  } while(e);
}
std::cout << x << std::endl;
```

Was berechnet diese Programm?

2^n

Was ist der Wertebereich für n?

$n \in [0 \dots 31]$

Zeige, das das Programm für alle $n \in [0 \dots 31]$ terminiert

Das Programm terminiert für $n = 0$ wegen dem if-Block.

Für $n \in [1 \dots 31], n \in \mathbb{N}$ ist $(\backslash k \backslash)$ streng monoton steigend mit Startwert 0.

Daher muss irgendwann $k = n$ gelten und das Programm terminiert.

Best of Vorlesung

Pre / Post Conditions

Precondition:

Nicht nötig

Postcondition:

```
// POST: return value is
//         the maximum of
//         i,j and k
```

```
double f (double i,
          double j,
          double k) {

    if (i > j) {
        if (i > k) {
            return i;
        }
        else{
            return k;
        }
    }
    else {
        if (j > k) {
            return j;
        }
        else {
            return k;
        }
    }
}
```

Pre / Post Conditions

```
double g (int i, int j) {
    double r = 0.0;
    for (int k = i; k <= j; ++k){
        r += 1.0 / k;
    }
    return r;
}
```

Precondition:

```
// PRE: 0 not contained
//       in {i, ..., j}
```

Postcondition:

```
// POST: return value is the sum
//        1/i + 1/(i+1) + ... + 1/j
```

Funktionen

Was ist der Output von diesem Programm, vorausgesetzt es gibt keine Overflows?

```
  i *  f(i)  *   f( f(i) )
= i *  (i*i) *   f( f(i) )
= i *  (i*i) *   f(  i*i )
= i *  (i*i) *  ((i*i)*(i*i))
```

$$= i^7$$

```cpp
#include <iostream>

int f (int i) {
    return i * i;
}

int g (int i) {
    return i * f(i) * f(f(i));
}

void h (int i) {
    std::cout << g(i) << "\n";
}

int main () {
    int i;
    std::cin >> i;
    h(i);
    return 0;
}
```

Perfect Numbers

Eine "Perfekte" Zahl ist eine Zahl, welche gleich der Summe all ihrer Teiler ist.

Beispiel:

1. $28 = 1 + 2 + 4 + 7 + 14$ ist perfekt
2. $12 < 1 + 2 + 3 + 4 + 6$ ist nicht perfekt.

"Skizziert" auf Papier eine Funktion, welches Zählt, wie viele dieser Perfekten Zahlen in einem Intervall [a,b] existieren.

```cpp
#include <iostream>
#include "perfect.h"

bool is_perfect(unsigned int number) {
    // [...]
}

unsigned int count_perfect( unsigned int a,
                            unsigned int b) {
    // [...]
}

int main () {
    // input
    unsigned int a;
    unsigned int b;
    std::cin >> a >> b;

    // computation and output
    unsigned int count = count_perfect(a, b);

    // output
    std::cout << count << std::endl;

    return 0;
}
```

Perfect Numbers

```cpp
#include <iostream>
#include "perfect.h"

bool is_perfect(unsigned int number) {
    unsigned int sum = 0;
    for (unsigned int d = 1; d < number; ++d) {
        if (number % d == 0) {
            sum += d;
        }
    }
    return sum == number;
}

unsigned int count_perfect( unsigned int a,
                            unsigned int b) {
  unsigned int count = 0;
  for (unsigned int i = a; i <= b; ++i) {
    if (is_perfect(i)) {
      std::cerr << i << std::endl;
      count++;
    }
  }
  return count;
}
```

Stepwise Refinment

Grosse Aufgaben von Grund auf zu programmieren kann häufig überwältigend wirken

- Wo soll man starten?
- Was für Funktionen brauchen wir?
- Wie soll man das Programm strukturieren?

Diese grossen Aufgaben können in kleine überschaubare Probleme aufgeteilt werden. → **stepwise approach**

Stepwise Refinment

Vorgehensweise

1. **Gliederung**
   Grobe Struktur mit Kommentaren
2. **Verfeinern**
   1. Genauere Kommentare
   2. Programmieren
   3. (Hypothetische) Funktionsaufrufe

Program of the Week

Stepwise Refinment

```cpp
int main(){
        //The number of sticks at the start of the game
        unsigned int sticksLeft = 17;

        //Indicates wether the computer can start or not.
        bool computersTurn = true;


        while(sticksLeft > 0){
                // Display game state
                printSticks(sticksLeft);

                if(computersTurn){ // Computer plays
                        unsigned int amount =  computerPlayer(sticksLeft);
                        std::cout << "The computer takes " << amount << " sticks." << std::endl;
                        sticksLeft -= amount;
                }
                else{ // Human plays
                        unsigned int amount =  humanPlayer(sticksLeft);
                        std::cout << "You take " << amount << " sticks." << std::endl;
                        sticksLeft -= amount;
                }
                //Switch players
                computersTurn = !computersTurn;
        }


        if(computersTurn){
                std::cout << "The computer won, too bad." << std::endl;
        }
        else{
                std::cout << "You won, congratulations!" << std::endl;
        }

        return 0;
}
```

1. **Gliederung**
   Grobe Struktur mit Kommentaren
2. **Verfeinern**
   1. Genauere Kommentare
   2. Programmieren
   3. **(Hypothetische)** Funktionsaufrufe

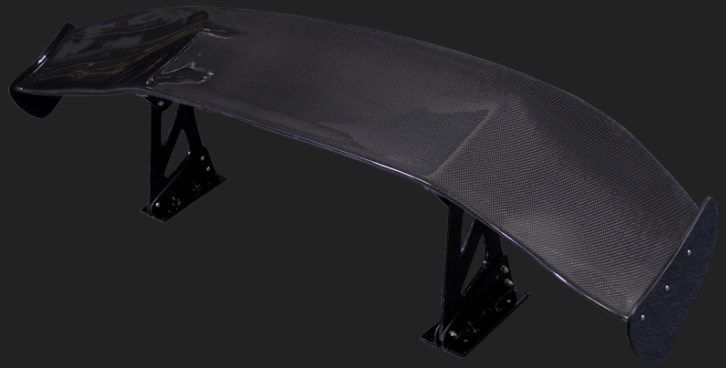
Stepwise Refinment

```
//PRE:  Number of sticks that are left on the field
//POST: the number of sticks n the user wants to take.
//      1 >= n >= 3
int humanPlayer(int sticksLeft){
        std::cout << "How many sticks would you like to take? ";
        int nSticks;

        do {
            std::cin>> nSticks;
            if(nSticks < 1 or nSticks > 3){
                std::cout << "Please enter a value between 1 and 3. ";
            }
            else if(nSticks > sticksLeft){
                std::cout << "There are only " << sticksLeft << " sticks left. ";
            }
        } while(nSticks < 1 or nSticks > 3 or nSticks > sticksLeft);

        return nSticks;
}
```

Stepwise Refinment

```
//PRE:  Number of sticks that are left on the field
//POST: the number of sticks n the computer wants to take.
//      1 >= n >= 3
int computerPlayer(int sticksLeft){
        int amount = (sticksLeft-1) % 4;

        if(amount == 0){
            amount = 1;
        }

        return amount;
}
```

Vorbesprechung



Pascal Schärli 22.03.2019

ETH zürich

1: Perpetual calender

Was für ein Wochentag war am 22.05.1996?

Schreibt ein Programm, welches zu einem beliebigen Datum den Wochentag herausfinden kann.

Tipps:

- Teilt die komplexe Aufgabe in kleine machbare Teilschritte auf. (Stepwise Refinment)
- Es gibt die Gaußsche Wochentagsformel, die darf man aber *nicht* verwenden.

2: Run-Length Encoding

```
0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9
```

$9 \cdot$ ”0”, $11 \cdot$ ”1”,
$11 \cdot$ ”2”, $9 \cdot$ ”3”,
$8 \cdot$ ”4”, $12 \cdot$ ”5”,
$12 \cdot$ ”6”, $8 \cdot$ ”7”,
$7 \cdot$ ”8”, $13 \cdot$ ”9”

```
 9  0 11  1
11  2  9  3
 8  4 12  5
12  6  8  7
 7  8 13  9
```

In dieser Aufgabe bauen wir uns einen Algorithmus um Byte-Sequenzen zu Komprimieren

2: Run-Length Encoding Tipps

```
0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9
```

```
 9  0 11  1
11  2  9  3
 8  4 12  5
12  6  8  7
 7  8 13  9
```

Decode:
Wiederhole bis -1:

1. Lese Anzahl Wiederholungen n
2. Lese Wert b
3. Gib Wert b genau n-mal aus

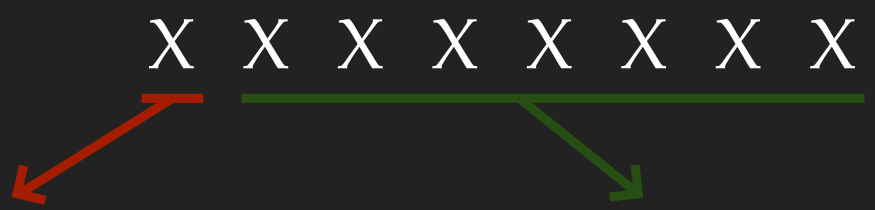
```
// POST: returns true if 0 <= value <= 255, otherwise false
bool is_byte(int value){/*...*/}

// PRE:  1 <= runlength <= 255, and value is a byte.
// POST: outputs run length encoded bytes of tuple
void output(unsigned int run_length,
            unsigned int value){/*...*/}

// POST: reads byte sequence and outputs encoded bytes
void encode(){/*...*/}


// POST: reads byte sequence and outputs decoded bytes
void decode(){/*...*/}
```

Encode:
Wiederhole bis -1:

1. Setze Counter c auf 1
2. Lies Wert b
3. Lies so lange b, bis anderer Wert kommt & erhöhe Counter
4. Output: *Counter Wert*

3: Run-Length Encoding Reloaded

Für Bytes mit run-length 1 (keine Wiederholung) ist der Algorithmus sehr ineffizient.



0: Einzelner Wert

1: Run-Length

Wert / Run-Length

3: Run-Length Encoding Reloaded

Sequenz Dezimal

```
0 0 1 2
```

Sequenz Binär

```
00000000
00000000
00000001
00000010
```

Encoding Binär

```
10000020
00000000
00000001
00000010
```

Encoding Dezimal

```
130 1 2
```

Sequenz Dezimal

```
200 20 20 20
```

Sequenz Binär

```
11001000
00010100
00010100
00010100
```

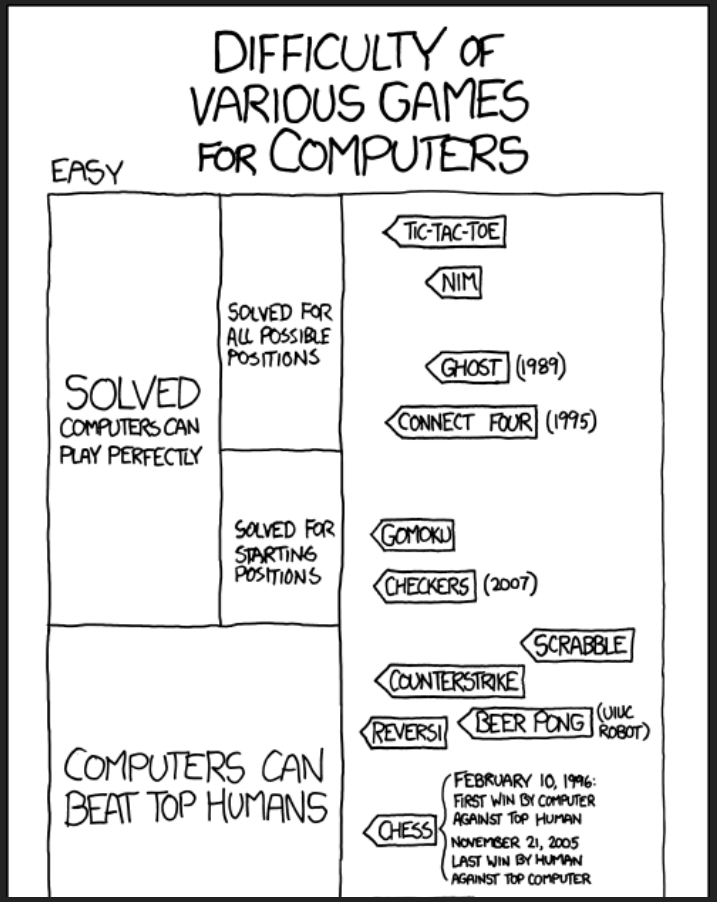
Encoding Binär

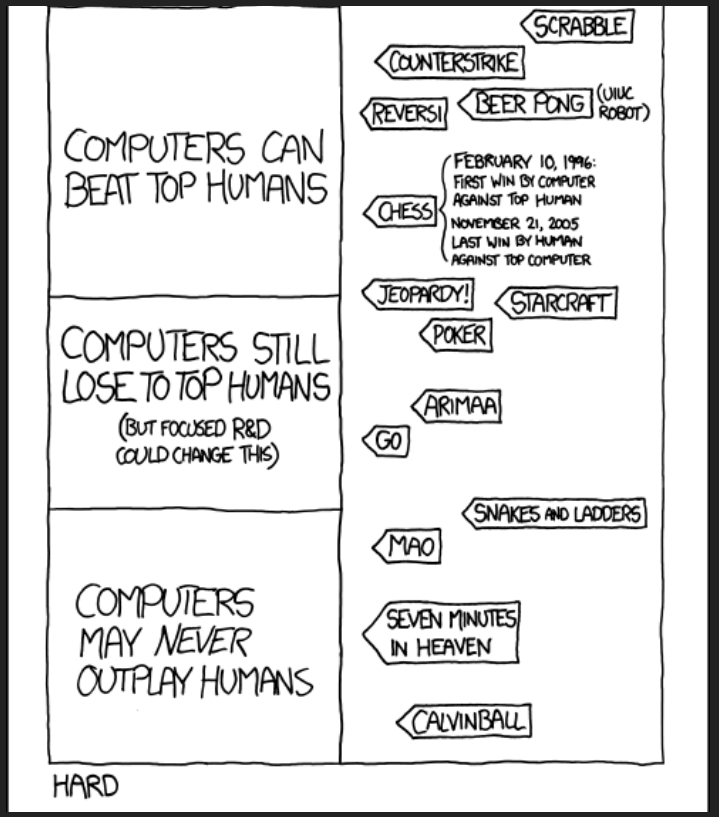
```
10000001
11001000
10000011
00010100
```

Encoding Dezimal

```
129 200 131 20
```

Viel Spass!


DIFFICULTY OF VARIOUS GAMES FOR COMPUTERS
EASY
SOLVED
COMPUTERS CAN PLAY PERFECTLY
SOLVED FOR ALL POSSIBLE POSITIONS
SOLVED FOR STARTING POSITIONS
COMPUTERS CAN BEAT TOP HUMANS
TIC-TAC-TOE
NIM
GHOST (1989)
CONNECT FOUR (1995)
GOMOKU
CHECKERS (2007)
SCRABBLE
COUNTERSTRIKE
REVERSI
BEER PONG (UIUC ROBOT)
CHESS
FEBRUARY 10, 1996: FIRST WIN BY COMPUTER AGAINST TOP HUMAN
NOVEMBER 21, 2005 LAST WIN BY HUMAN AGAINST TOP COMPUTER


COMPUTERS CAN BEAT TOP HUMANS
COMPUTERS STILL LOSE TO TOP HUMANS
(BUT FOCUSED R&D COULD CHANGE THIS)
COMPUTERS MAY NEVER OUTPLAY HUMANS
HARD
SCRABBLE
COUNTERSTRIKE
REVERSI
BEER PONG (UIUC ROBOT)
CHESS
FEBRUARY 10, 1996: FIRST WIN BY COMPUTER AGAINST TOP HUMAN
NOVEMBER 21, 2005 LAST WIN BY HUMAN AGAINST TOP COMPUTER
JEOPARDY!
STARCRAFT
POKER
ARIMAA
GO
SNAKES AND LADDERS
MAO
SEVEN MINUTES IN HEAVEN
CALVINBALL