ETH zürich

Übungsstunde W03

Computer Science (CSE) – AS 23

Übersicht

Heutiges Programm

Follow-up
Feedback zu **code** expert
Boolean Expressions
`for`-Schleifen
Simple Manual Debugging
Tipps zu **code** expert
Outro

`n.ethz.ch/~agavranovic`

1. Follow-up

Follow-up aus vorherigen Übungsstunden

Follow-up aus vorherigen Übungsstunden

- Two's complement
- Alternative Basen
- L- vs. R-Values
- Autograder
- Deutsch oder Englisch?
- `0b`, `0x`, `u` (ganz am Schluss)

Alternative Basen

Zahlen in anderen Basen funktionieren konzeptionell genau gleich

Alternative Basen

Zahlen in anderen Basen funktionieren konzeptionell genau gleich

binär	hexadezimal	dezimal
1111	F	15
1’1111	1F	31
11’0111’1100’0101	37C5	14’277
1010’1100’1101’1100	ACDC	44’252
1’0000’0000’0000’0000	1’0000	65’536
1010’1111’1111’1110’0000’1000’0001’0101	AFFE’0815	2’952’661’013

Fragen/Unklarheiten?

Two’s complement

Konversion von Dezimahl zu Binär:

1. Nehme den Betrag der Zahl und schreibe die Binärrepräsentation davon
2. Bits flippen: 1 wird zu 0 und 0 wird zu 1
3. 1 addieren und den Overflow ignorieren

Two's complement

Konversion von Dezimahl zu Binär:

1. Nehme den Betrag der Zahl und schreibe die Binärrepräsentation davon
2. Bits flippen: 1 wird zu 0 und 0 wird zu 1
3. 1 addieren und den Overflow ignorieren

Beispiel: -6

1. $+6$ lässt sich als 0110 schreiben
2. Nach Bit Flipping erhalten wir 1001
3. $1001 + 1 = 1010$

Two's complement

Tipps zu Two's Complement:

- Das erste Bit einer Zahl n heisst **most significant bit** und gibt das Vorzeichen an

$$\text{msb} \begin{cases} = 1 \rightarrow n < 0 \\ = 0 \rightarrow n >= 0 \end{cases}$$

- Die Grösse (= Anzahl Bits) einer Zahl bestimmt, welche Zahlen dargestellt werden können
  - Mit n Bits können Zahlen von $-(2^{n-1})$ bis $2^{n-1} - 1$ dargestellt werden

Fragen/Unklarheiten?

L- vs. R-Values

- **L-Value:** Ein Objekt mit identifizierbarer Adresse
  - Beispiele

    ```
    a = 9
    a = b
    ```

- **R-Value:** Ein Objekt ohne Adresse; alles, was nicht ein L-Value ist
  - Beispiele

    ```
    a + b
    3
    ```

L- vs. R-Values

Aufgabe

Sind die folgenden Expressions L- oder R-Values?

1. `(a * b) + c`
2. `(x = y = 4)`
3. `188`
4. `25 * z = a / b`

L- vs. R-Values

Aufgabe

Sind die folgenden Expressions L- oder R-Values?

1. `(a * b) + c`
2. `(x = y = 4)`
3. `188`
4. `25 * z = a / b`

Lösung

1. **R-Value**: Expression ist eine Addition von zwei Variablen, was ein R-Value ergibt (hat keine Adresse)
2. **L-Value**: `x` hat eine Adresse
3. **R-Value**: `188` hat keine Adresse
4. **invalid**: `25 * z` ist ein R-Value, eine Zuweisung zu einem R-Value ist nicht möglich.

Fragen/Unklarheiten?

Autograder

- Benutzt den Autograder als Hilfsmittel und testet Lösungen mit ihm

Deutsch oder Englisch?

- Werden im Verlauf der Woche eine (anonyme) Abstimmung dazu abhalten, ob die Übungsstunde in Zukunft auf Deutsch oder Englisch gehalten werden soll

2. Feedback zu **code** expert

Allgemeines zu **code** expert

Allgemeines zu **code** expert

- Noch nicht durch mit den Korrekturen…
- Ihr werdet alle im Laufe der Woche noch persönliches Feedback bekommen

Fragen zu **code** expert eurerseits?

Ziele

Ziele für heute

- ☐ *boolean expressions* richtig evaluieren
- ☐ wissen, was *short circuiting* ist und wo es als Fehlerquelle vorkommt
- ☐ Programme mit for-Schleifen *tracen* können
- ☐ Einfache Programme, die for-Schleifen enthalten, schreiben können

3. Boolean Expressions

Boolean Expressions

- `bool` ist ein Datentyp der nur zwei Werte annehmen kann
  - `true`
  - `false`
- (Klammern) sind eure besties
- die Reihenfolge ist beim Auswerten von Bedeutung (Links nach rechts und *Operator Precedence*)
- Vorsicht vor *short circuits*

Booleans

- kurz und einfach: `bool`s
- immer entweder `true` oder `false`

Booleans

- kurz und einfach: `bool`s
- immer entweder `true` oder `false`
- wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein

Booleans

- kurz und einfach: `bool`s
- immer entweder `true` oder `false`
- wenn `true` in eine Zahl umgewandelt wird,
  wird es die Zahl 1 sein
- wenn eine Zahl $\neq 0$ in einen `bool` umgewandelt wird,
  wird es der Wert `true` sein

Booleans

- kurz und einfach: `bool`s
- immer entweder `true` oder `false`
- wenn `true` in eine Zahl umgewandelt wird,
  wird es die Zahl 1 sein
- wenn eine Zahl $\neq 0$ in einen `bool` umgewandelt wird,
  wird es der Wert `true` sein
- wenn `false` in eine Zahl umgewandelt wird,
  wird es die Zahl 0 sein

Booleans

- kurz und einfach: `bool`s
- immer entweder `true` oder `false`
- wenn `true` in eine Zahl umgewandelt wird,
  wird es die Zahl 1 sein
- wenn eine Zahl $\neq 0$ in einen `bool` umgewandelt wird,
  wird es der Wert `true` sein
- wenn `false` in eine Zahl umgewandelt wird,
  wird es die Zahl 0 sein
- wenn 0 in einen `bool` umgewandelt wird,
  wird es der Wert `false` sein

Precedence Ranking

[1] Gezeigt sind hier nur die z. Z. wichtigsten. Für den Rest, siehe cppreference

Precedence Ranking

Precedence Ranking[1]

1. `a++`, `a--`
2. `++a`, `--a`, `-a`, `!a`, `*a`, `&a`
3. `*`, `/`, `%`
4. `+`, `-`
5. `<`, `<=`, `>`, `>=`
6. `==` `!=`
7. `&&`
8. `||`
9. `=`, `+=`, `-=`, `*=`, `/=`, `%=`

[1] Gezeigt sind hier nur die z. Z. wichtigsten. Für den Rest, siehe cppreference

(Nutzt) (Klammern)

- Klammern funktionieren wie in der Mathematik
- sie werden oft gebraucht, um die korrekte Evaluation offensichtlich zu machen
- …oder eben um die Evaluationsreihenfolge abzuändern

(Nutzt) (Klammern)

- Klammern funktionieren wie in der Mathematik
- sie werden oft gebraucht, um die korrekte Evaluation offensichtlich zu machen
- …oder eben um die Evaluationsreihenfolge abzuändern

Aufgabe

Mache die Evaluation mittels Klammern offensichtlich:

```
3 < 4 + 1 && 2 < 3
```

Tipp: nutze die vorherige Folie

(Nutzt) (Klammern)

- Klammern funktionieren wie in der Mathematik
- sie werden oft gebraucht, um die korrekte Evaluation offensichtlich zu machen
- …oder eben um die Evaluationsreihenfolge abzuändern

Aufgabe

Mache die Evaluation mittels Klammern offensichtlich:

```
3 < 4 + 1 && 2 < 3
```

Tipp: nutze die vorherige Folie

Lösung

```
(3 < (4 + 1)) && (2 < 3)
```

Mehrere Operatoren mit gleicher Präzedenz

Assoziativität

- Sagt aus, in welcher Reihenfolge Operationen evaluiert werden
- Entweder Right-to-left ($\leftarrow$) oder Left-to-right ($\rightarrow$)
- *Am besten eine Tabelle auf die Zusammenfassung mit Präzedenz und Assoziativität schreiben*

Mehrere Operatoren mit gleicher Präzedenz

Kurzaufgabe

Wie würdest du klammern, damit die Evaluation der unteren Expression offensichtlich wird?
Tipp: Die Assoziativität von ***&&*** *ist Left-to-right*

```
a && b && c
```

Mehrere Operatoren mit gleicher Präzedenz

Kurzaufgabe

Wie würdest du klammern, damit die Evaluation der unteren Expression offensichtlich wird?
Tipp: Die Assoziativität von `&&` *ist Left-to-right*

```
a && b && c
```

Lösung

Einfach von links nach rechts:

```
(a && b) && c
```

Short Circuits

Short Circuit

Short Circuits

Short Circuit

Die bool'schen Operatoren **`&&`** und **`||`** evaluieren zuerst die linke Expression und dann *nur falls nötig* die rechte Expression.

Insbesondere heisst das, dass die rechte Expression *nicht* evaluiert wird, wenn man das Ergebnis der gesamten Evaluation bereits herleiten kann.

Short Circuits in Code

```
if(3 > 2 && 10 > 11){
    std::cout << "Of course not!\n";
}    // not a short circuit evaluation
```

Short Circuits in Code

```
int a = 3;

if(false && ++a < 2){
    std::cout << "Of course not!\n";
} // a short circuit evaluation

std::cout << a << "\n"; // what will be the output?

if(++a < 2 && false){
    std::cout << "Of course not!\n";
} // another short circuit evaluation

std::cout << a << "\n"; // what will be the output?
```

Verständniskontrolle

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
x == 1 || 1 / (x - 1) < 1
```

Vergesst nicht: Klammern sind eure Besties

Verständniskontrolle

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
x == 1 || 1 / (x - 1) < 1
```

Vergesst nicht: Klammern sind eure Besties

Lösung

Zuerst: Klammern!

```
(x == 1) || ((1 / (x - 1)) < 1)
```

beginne links

Verständniskontrolle

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
x == 1 || 1 / (x - 1) < 1
```

Vergesst nicht: Klammern sind eure Besties

Lösung

Zuerst: Klammern!

```
(x == 1) || ((1 / (x - 1)) < 1)
```

beginne links

```
(1 == 1) || ((1 / (x - 1)) < 1)
```

Verständniskontrolle

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
x == 1 || 1 / (x - 1) < 1
```

Vergesst nicht: Klammern sind eure Besties

Lösung

Zuerst: Klammern!

```
(x == 1) || ((1 / (x - 1)) < 1)
```

beginne links

```
(1 == 1) || ((1 / (x - 1)) < 1)
true || ((1 / (x - 1)) < 1)
```

Verständniskontrolle

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
x == 1 || 1 / (x - 1) < 1
```

Vergesst nicht: Klammern sind eure Besties

Lösung

Zuerst: Klammern!

```
(x == 1) || ((1 / (x - 1)) < 1)
```

beginne links

```
(1 == 1) || ((1 / (x - 1)) < 1)
true || ((1 / (x - 1)) < 1)
true
```

(**Merke:** `true || ...`) evaluiert immer zu (`true`)

Und noch eins

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

Und noch eins

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

Lösung

```
!(1 < 2 && x == 1) + 1
```

Und noch eins

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

Lösung

```
!(1 < 2 && x == 1) + 1
(!((1 < 2) && (x == 1))) + 1
```

Und noch eins

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

Lösung

```
!(1 < 2 && x == 1) + 1
(!((1 < 2) && (x == 1))) + 1
(!((true) && (true))) + 1
```

Und noch eins

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

Lösung

```
!(1 < 2 && x == 1) + 1
(!((1 < 2) && (x == 1))) + 1
(!((true) && (true))) + 1
(!(true)) + 1
```

Und noch eins

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

Lösung

```
!(1 < 2 && x == 1) + 1
(!((1 < 2) && (x == 1))) + 1
(!((true) && (true))) + 1
(!(true)) + 1
false + 1
```

Und noch eins

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

Lösung

```
!(1 < 2 && x == 1) + 1
(!((1 < 2) && (x == 1))) + 1
(!((true) && (true))) + 1
(!(true)) + 1
false + 1
0 + 1
```

Und noch eins

Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

Lösung

```
!(1 < 2 && x == 1) + 1
(!((1 < 2) && (x == 1))) + 1
(!((true) && (true))) + 1
(!(true)) + 1
false + 1
0 + 1
1
```

Fragen/Unklarheiten?

4. `for`-Schleifen

for-Schleifen

- Was ist eine Schleife
- Kurzeinführung zu *Scopes*
- Kurzeinführung ins *Program Tracing*

Kurzeinführung zu Scopes

- Praktisch immer, wenn Ihr {diese geschweiften Klammern} verwendet, erstellt ihr einen {Scope}

Kurzeinführung zu Scopes

- Praktisch immer, wenn Ihr {diese geschweiften Klammern} verwendet, erstellt ihr einen {Scope}
- Stellt euch einen Scope als eine "Welt in einer Welt" vor

Kurzeinführung zu Scopes

- Praktisch immer, wenn Ihr {diese geschweiften Klammern} verwendet, erstellt ihr einen {Scope}
- Stellt euch einen Scope als eine "Welt in einer Welt" vor
- Informationen/Variablen können nicht aus einem Scope rausfliessen, aber Informationen/Variablen von aussen sind im inneren Scope verfügbar

Kurzeinführung zu Scopes

- Praktisch immer, wenn Ihr {diese geschweiften Klammern} verwendet, erstellt ihr einen {Scope}
- Stellt euch einen Scope als eine "Welt in einer Welt" vor
- Informationen/Variablen können nicht aus einem Scope rausfliessen, aber Informationen/Variablen von aussen sind im inneren Scope verfügbar
- Wenn das Programm bei der rechten geschweiften Klammer des Scopes ankommt, stirbt jegliche Information/Variable innerhalb dieses Scopes

Allgemeine Struktur einer for-Schleife

```
for(init; condition; expression){
    statement 1;
    statement 2;
    ...
}
```

Wichtige Bemerkung

Der **`expression`**-Teil wird *nach* den **`statements`** ausgeführt.

Program Tracing

"*Program Tracing* ist der
Prozess des Ausführens eines Programms von Hand mit konkreten Eingaben."

Ziemlich wichtiger Skill, insbesondere am Anfang. Irgendwann werdet ihr das quasi im Kopf können. Einen ausführlichen Guide dazu findet ihr hier:

▸ Program Tracing

`for` Loop

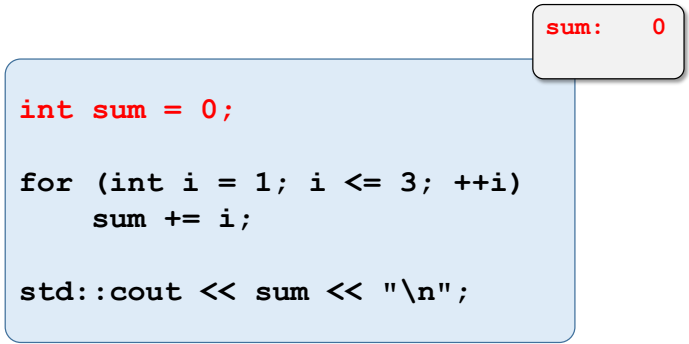
Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop



Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    0
i:      1
```

Example – `for` Loop

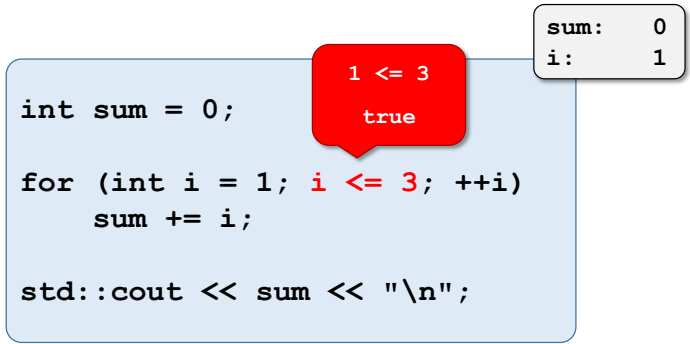
```
sum:    0
i:      1
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop



Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    1
i:      1
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    1
i:      2
```

Example – `for` Loop

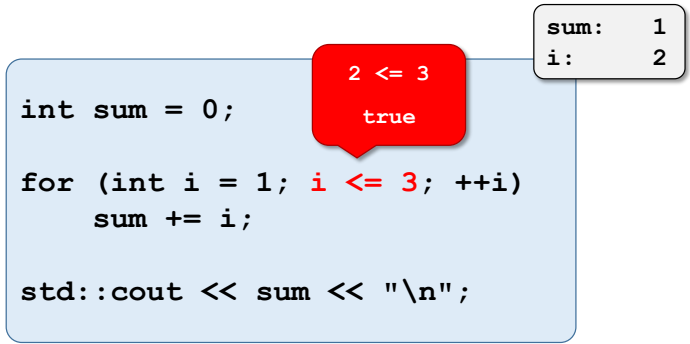
```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    1
i:      2
```

Example – `for` Loop



Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    3
i:      2
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    3
i:      3
```

Example – `for` Loop

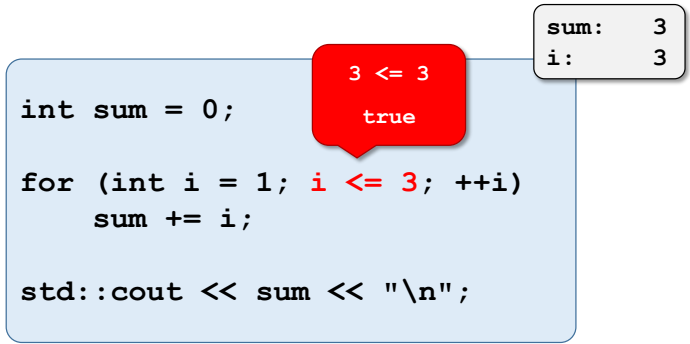
```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    3
i:      3
```

Example – `for` Loop



Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    6
i:      3
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    6
i:      4
```

Example – `for` Loop

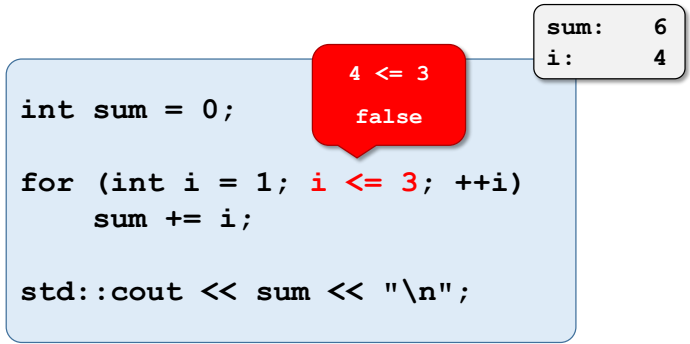
```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    6
i:      4
```

Example – `for` Loop



Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    6
```

Fragen/Unklarheiten?

Aufgabe *Strange Sum*

Aufgabe

Öffnet *Strange Sum* auf **code** expert und versucht es zuerst mit Stift und Papier zu lösen. (10min)

Description

Write a program that reads a number $n > 0$ from standard input and outputs the sum of all positive numbers up to n that are odd but not divisible by 5.

Aufgabe *Strange Sum*

Aufgabe

Öffnet *Strange Sum* auf **code** expert und versucht es zuerst mit Stift und Papier zu lösen. (10min)

Description

Write a program that reads a number $n > 0$ from standard input and outputs the sum of all positive numbers up to n that are odd but not divisible by 5.

Aufgabe

Und jetzt schreibt das dazugehörige Programm. (5min)

Fragen/Unklarheiten?

Mögliche Lösung zu *Strange Sum*

```cpp
// input
unsigned int strangesum = 0;
unsigned int n;
std::cin >> n;

// computation
for(unsigned int i = 1; i <= n; i++){
    if((i % 2) == 1){
        if(i % 5){
            strangesum += i;
        }
    }
}

// output
std::cout << strangesum << "\n";
```

Kompaktere Lösung zu *Strange Sum*

```
// input
unsigned int strangesum = 0;
unsigned int n;
std::cin >> n;

// computation
for(unsigned int i = 1; i <= n; i++){
    if( ((i % 2) == 1) && (i % 5) ){
        strangesum += i;
    }
}

// output
std::cout << strangesum << "\n";
```

Noch kompaktere Lösung zu *Strange Sum*

```
// input
unsigned int strangesum = 0;
unsigned int n;
std::cin >> n;

// computation
for(unsigned int i = 1; i <= n; i+=2){
    if(i % 5){
        strangesum += i;
    }
}

// output
std::cout << strangesum << "\n";
```

Aufgabe *Largest Power*

Aufgabe

Öffnet *Largest Power* auf **code** expert und versucht es zuerst mit Stift und Papier zu lösen. (10min)

Description

Write a program that inputs a positive natural number n and outputs the largest number p that is a power of 2 and smaller or equal to n.

Aufgabe *Largest Power*

Aufgabe

Öffnet *Largest Power* auf **code** expert und versucht es zuerst mit Stift und Papier zu lösen. (10min)

Description

Write a program that inputs a positive natural number n and outputs the largest number p that is a power of 2 and smaller or equal to n.

Aufgabe

Und jetzt schreibt das dazugehörige Programm. (5min)

Aufgabe *Largest Power*

Aufgabe

Öffnet *Largest Power* auf **code** expert und versucht es zuerst mit Stift und Papier zu lösen. (10min)

Description

Write a program that inputs a positive natural number n and outputs the largest number p that is a power of 2 and smaller or equal to n.

Aufgabe

Und jetzt schreibt das dazugehörige Programm. (5min)

Aufgabe

Besprecht eure Ansätze mit den Personen neben euch. Hattet ihr den gleichen Ansatz? Was könnt ihr voneinander lernen? (7min)

Mögliche Lösung zu *Largest Power*

```cpp
#include <iostream>
#include <cassert>

int main () {
  unsigned int n;
  std::cin >> n;
  assert(n >= 1);

  unsigned int power = 1;
  for (; power <= n / 2; power *= 2);

  std::cout << power << std::endl;

  return 0;
}
```

Fragen/Unklarheiten?

5. Simple Manual Debugging

Debugging

Debugging

…is the process of finding and resolving bugs (defects or problems that prevent correct operation) within programs, software, or systems.

Debugging

Debugging

…is the process of finding and resolving bugs (defects or problems that prevent correct operation) within programs, software, or systems.

Debugging

```
int main () {
    const int n = 6;

    // Compute n^12
    int prod = 1;
    for (int i = 1; 1 <= i < 13; ++i) {
        prod *= n;
    }

    // Output stars
    for (int i = 1; i < prod; ++i) {
        std::cout << "*";
    }
    std::cout << "\n";
    return 0;
}
```

Debugging

Frage

Wie könnten wir herausfinden, bei welcher Zeile das Programm feststeckt?

Debugging

Frage

Wie könnten wir herausfinden, bei welcher Zeile das Programm feststeckt?

Antwort

Sachen ausgeben lassen an interessanten Stellen im Code

Debugging

Frage

Wie könnten wir herausfinden, bei welcher Zeile das Programm feststeckt?

Antwort

Sachen ausgeben lassen an interessanten Stellen im Code

Frage

Wieso steckt das Programm in der ersten for-Schleife fest?

Debugging

Frage

Wie könnten wir herausfinden, bei welcher Zeile das Programm feststeckt?

Antwort

Sachen ausgeben lassen an interessanten Stellen im Code

Frage

Wieso steckt das Programm in der ersten for-Schleife fest?

Antwort

Die condition ist falsch geschrieben

Debugging

Frage

Wie könnten wir herausfinden, bei welcher Zeile das Programm feststeckt?

Antwort

Sachen ausgeben lassen an interessanten Stellen im Code

Frage

Wieso steckt das Programm in der ersten for-Schleife fest?

Antwort

Die condition ist falsch geschrieben
Sollte sein: `1 <= i && i < 13`.

Debugging

Frage

Wie können wir ermitteln, wieso trotzdem nichts ausgegeben wird?

Debugging

Frage

Wie können wir ermitteln, wieso trotzdem nichts ausgegeben wird?

Antwort

Einfach den Wert von **`prod`** nach der ersten Schleife ausgeben lassen

Debugging

Frage

Wie können wir ermitteln, wieso trotzdem nichts ausgegeben wird?

Antwort

Einfach den Wert von **`prod`** nach der ersten Schleife ausgeben lassen

Frage

Wie finden wir raus, wieso **`prod`** negativ wurde?

Debugging

Frage

Wie können wir ermitteln, wieso trotzdem nichts ausgegeben wird?

Antwort

Einfach den Wert von **`prod`** nach der ersten Schleife ausgeben lassen

Frage

Wie finden wir raus, wieso **`prod`** negativ wurde?

Antwort

Einfach den Wert von **`prod`** in *jeder* Iteration in der Schleife ausgeben lassen

Fragen/Unklarheiten?

6. Tipps zu **code** expert

Tipps für **code** expert

Task 1.5: “two-complement integer representation (Optional)”

- nichts $\rightarrow$ Basis 10
- `0b` $\rightarrow$ Basis 2
- `0` $\rightarrow$ Basis 8 (also `10` $\neq$ `010`)
- `0x` $\rightarrow$ Basis 16

Task 3: “From decimal to binary representation”

- Denkt an letzte Übungsstunde…

Task 4b: “Fibonacci overflow check”

- ***Most importantly***: *exit the print loop as soon as you detect that an overflow would occur.*, also darf die Addition *nicht* passieren!

7. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche!