ETH Zürich

Informatik I
Übungsstunde 06

Johannes Odenthal

cs.jodenthal.com

Hausaufgabenfragen

Welche Fragen sind bei der letzten Übung offen geblieben?

Hausaufgabenreview · Exercise 05

is_valid_date: Hilfsfunktion nutzen

Don’t · Ausschnitt

```
switch (month) {
  case 2: days = 28;
    // ... Schaltjahr
    break;
  // ... weitere Monate
}
return 0 < day
    && day <= days;
```

Do · Ausschnitt

```
return 0 < day
  && day <=
  count_days_in_month(
    month, year);
```

is_even: Alle Pfade beachten

Don’t

```
bool is_even(int i) {
  if (i % 2 == 0) {
    return true;
  }
}
```

Do

```
bool is_even(int i) {
  return i % 2 == 0;
}
```

decode: Endmarkierung prüfen

Don’t · Ausschnitt

```
std::cin >> count
         >> value;
if (count == -1) {
  break;
}
```

Do · Ausschnitt

```
std::cin >> count;
if (count == -1) {
  break;
}
std::cin >> value;
```

Neu: Feedback Opt-Out

Ihr braucht oder lest mein Feedback nicht?

Schreibt bitte oben in eure Abgabedateien:

```
// no-feedback
```

References

Referenz als Alias

```
1 int main() {
2   int a = 3;
3   int& b = a;
4   b = 7;
5   std::cout << a << "\n";
6 }
```

Was wird ausgegeben?

Code Tracing

`a = 3`

`b` ist ein Alias für `a`.

`b = 7` verändert auch `a`.

Ausgabe: `7`

Pass by value

```
1 void foo(int i) {
2   i = 5;
3 }
4
5 int main() {
6   int i = 4;
7   foo(i);
8   std::cout << i << "\n";
9 }
```

Was wird ausgegeben?

Code Tracing

`main`: `i = 4`.

`foo`: neue Kopie mit Wert 4.

`i = 5`: nur die Kopie ändert sich.

`foo` endet: die Kopie endet; das Original bleibt 4.

Ausgabe: `4`.

Pass by reference

```
1 void foo(int& i) {
2   i = 5;
3 }
4
5 int main() {
6   int i = 4;
7   foo(i);
8   std::cout << i << "\n";
9 }
```

Was wird ausgegeben?

Code Tracing

`main`: `i = 4`.

`foo`: Alias für das `i` in `main`.

`i = 5`: das Original ändert sich.

`foo` endet: das Original ist weiterhin 5.

Ausgabe: `5`.

Zuweisen ist kein Neubinden

```cpp
int main() {
  int a = 3;
  int c = 9;
  int& b = a;
  b = c;
  b = 7;
  std::cout << a << " " << b << " " << c;
}
```

`7 7 9`

`b = c` weist den Wert zu. `b` bleibt Alias für `a`.

Lesen ohne Kopie

```cpp
#include <vector>

void print_first(const std::vector<int>& v) {
  if (!v.empty()) {
    std::cout << v[0] << "\n";
  }
}

int main() {
  std::vector<int> v{3, 5};
  print_first(v);
}
```

`&`: keine Vektorkopie.

`const`: kein Schreiben über `v`. Ausgabe: `3`.

Referenz als Rückgabewert

```cpp
int& increment(int& m) {
  return ++m;
}

int main() {
  int n = 3;
  increment(increment(n));
  std::cout << n << "\n";
}
```

Was wird ausgegeben?

Code Tracing

Innerer Aufruf: `n` wird 4. Rückgabe: Alias auf `n`.

Äusserer Aufruf: derselbe Alias; `n` wird 5.

Ausgabe: `5`.

Beispiel: Returning References

```cpp
#include <vector>

int& first(std::vector<int>& v) {
  return v[0];
}

int main() {
  std::vector<int> v{3, 5, 7};
  std::cout << first(v) << "\n";
  first(v) = 1;
  std::cout << first(v) << "\n";
}
```

Was macht die Funktion first?

Pre-Condition: `v` ist nicht leer.

Code Tracing

Erstes Lesen: Ausgabe `3`.

`first(v)` liefert einen Alias auf `v[0]`.

`first(v) = 1` verändert `v`: `1 5 7`.

Zweites Lesen: Ausgabe `1`.

References · Exercise 1

```cpp
int main() {
  int a = 0;
  int b = 1;
  for (int i = 0; i < 5; ++i) {
    b = foo(a, b);
    std::cout << b << " ";
  }
  return 0;
}
```

```cpp
int foo(int& a, int b) {
  a += b;
  return a;
}
```

Was wird ausgegeben?

Mit Papier überlegen:

(a) `int& a, int b`

(b) `int a, int b`

(c) `int a, int& b`

Für jede Variante neu mit `a = 0`, `b = 1` beginnen.

Exercise 1 (a)

```cpp
int foo(int& a, int b) {
  a += b;
  return a;
}

int main() {
  int a = 0;
  int b = 1;
  for (int i = 0; i < 5; ++i) {
    b = foo(a, b);
    std::cout << b << " ";
  }
  return 0;
}
```

Aufruf	Variablen	Ausgabe
1	`a = 1, b = 1`	`1`
2	`a = 2, b = 2`	`2`
3	`a = 4, b = 4`	`4`
4	`a = 8, b = 8`	`8`
5	`a = 16, b = 16`	`16`

Exercise 1 (b)

```cpp
int foo(int a, int b) {
  a += b;
  return a;
}

int main() {
  int a = 0;
  int b = 1;
  for (int i = 0; i < 5; ++i) {
    b = foo(a, b);
    std::cout << b << " ";
  }
  return 0;
}
```

Aufruf	Variablen	Ausgabe
1	`a = 0, b = 1`	`1`
2	`a = 0, b = 1`	`1`
3	`a = 0, b = 1`	`1`
4	`a = 0, b = 1`	`1`
5	`a = 0, b = 1`	`1`

Exercise 1 (c)

```cpp
int foo(int a, int& b) {
  a += b;
  return a;
}

int main() {
  int a = 0;
  int b = 1;
  for (int i = 0; i < 5; ++i) {
    b = foo(a, b);
    std::cout << b << " ";
  }
  return 0;
}
```

Aufruf	Variablen	Ausgabe
1	`a = 0, b = 1`	`1`
2	`a = 0, b = 1`	`1`
3	`a = 0, b = 1`	`1`
4	`a = 0, b = 1`	`1`
5	`a = 0, b = 1`	`1`

Reversing Vectors

CodeExpert

Fragen?

Multidimensional Vectors

Ein Vektor von int

```cpp
#include <vector>

int main() {
  std::vector<int> v(3, 0);
  v[2] = 7;
}
```

```
0  0  7
```

Ein Vektor enthält hier drei Werte vom Typ `int`.

Eine Matrix ist ein Vektor von Vektoren

```cpp
#include <vector>

int main() {
  std::vector<std::vector<int>> matrix(
      2, std::vector<int>(3, 0));
  matrix[1][2] = 7;
}
```

Zwei Zeilen, drei Spalten:
`0 0 0`
`0 0 7`
`matrix[Zeile][Spalte]`

$M_{i,j}$ entspricht `matrix[i - 1][j - 1]`.
C++ ab 0, Mathematik hier ab 1.

Matrix Transpose

CodeExpert

Fragen?

Hausaufgaben für nächste Woche

Week 6 · Task 3: [Hidden tests] Sorting by Swapping

Bei der Nachkorrektur werden die sichtbaren Tests ersetzt – wie in der Prüfung.

Eure Lösung muss auch mit anderen Eingaben funktionieren.

Optional · Nachschlagen

Optional · References, Exercise 2

Schreibt eine Funktion `swap`, die zwei `int`-Werte vertauscht.

```cpp
// Ergänzt swap vor main.
int main() {
  int a = 5;
  int b = 6;
  swap(a, b);
  std::cout << a << "\n" << b << "\n";
}
```

Gewünschte Ausgabe: `6`, dann `5`.

Exercise 2 · swap

```cpp
void swap(int& i, int& j) {
  const int h = i;
  i = j;
  j = h;
}

int main() {
  int a = 5;
  int b = 6;
  swap(a, b);
  std::cout << a << "\n"
      << b << "\n";
}
```

Code Tracing

`h = 5` bewahrt den alten Wert.

`i = j`: jetzt `a = 6`, `b = 6`.

`j = h`: jetzt `a = 6`, `b = 5`.

Ausgabe: `6`, dann `5`.

Optional · Mehr als ein Ergebnis

```
1 int solve_quadratic_equation(
2   const double a, const double b, const double c,
3   double& x1, double& x2);
```

Wie können zusätzlich zum Rückgabewert zwei Lösungen beim Aufrufer ankommen?

> Der Rückgabewert beschreibt die Anzahl der Lösungen.
>
> `x1` und `x2` sind Ausgabeparameter: Die Funktion schreibt vorhandene Lösungen in die Objekte des Aufrufers.

Nachschlagen · Referenzen und const

```
1 int a = 3;
2 int& b = a;
3 const int& c = a;
4 a = 7; // b und c lesen jetzt 7
5 const int& d = 5; // erlaubt
```

> Eine nicht-const Referenz braucht hier ein passendes veränderbares Objekt.
>
> Eine const-Referenz erlaubt Lesen, aber kein Schreiben über diese Referenz.

Bei `const int& d = 5` bleibt der direkt gebundene temporäre Wert bis zum Ende des Scopes erhalten.

Nachschlagen · Absichtlich ungültig

```
1 int& a = 3;           // Fehler
2 bool flag = false;
3 int& b = flag;        // Fehler
4 const int value = 4;
5 int& c = value;       // Fehler
6 const int& d = value;
7 d = 8;                // Fehler
```

Die Referenz braucht einen passenden veränderbaren Wert.

Über eine const-Referenz darf nicht geschrieben werden.

Nachschlagen · Vektoroperationen

```cpp
std::vector<int> v(3, 0); // 0 0 0
v[1] = 7;                 // 0 7 0
v.push_back(9);           // 0 7 0 9
auto n = std::ssize(v);   // 4
std::cout << v.at(3);     // 9
```

`at(i)` prüft die Grenze und wirft bei ungültigem Index `std::out_of_range`.

Für `v[i]` muss der Index gültig sein; eine Diagnose ist nicht garantiert.