ETH zürich

Übungsstunde W10

Informatik (RW & CBB & Statistik) – HS 23

Übersicht

Heutiges Programm

Follow-up
Ziele
Klassen und Operator-Überladung
Aufgabe "Tribool"
Iteratoren
Aufgabe "Find Max"
Outro

`rwko.ch/lily`

1. Follow-up

Follow-up

- PVKs: Was braucht ihr?
  - Fächer
  - Informationen

Fragen/Unklarheiten?

2. Ziele

Ziele

- ☐ eigene Klassen definieren können
- ☐ Operatoren für bereits definierte Klassen überladen können
- ☐ Iteratoren verwenden können

3. Klassen und Operator-Überladung

Funktionen voneinander differenzieren

Es ist möglich, dass zwei Funktionen den gleichen Namen haben, solange der Compiler eine andere Möglichkeit hat, sie zu unterscheiden. Die einzigen möglichen Kriterien Funktionen zu unterscheiden sind also:

- Namen der Funktionen
- Anzahl der Funktionsargumente
- Typen der Funktionsargumente

Putting the *Fun* in *Function* I

Wird dies zu einem Compilerfehler führen?

```
int fun1(const int a){
    // ...
}

int fun1(const int a, const int b){
    // ...
}
```

Putting the *Fun* in *Function* I

Wird dies zu einem Compilerfehler führen?

```
int fun1(const int a){
    // ...
}

int fun1(const int a, const int b){
    // ...
}
```

Antwort: Nein, weil die beiden Funktionen unterschiedlich viele Argumente besitzen (1 vs 2)

Putting the *Fun* in *Function* II

Wird dies zu einem Compilerfehler führen?

```
int fun2(const int a){
    // ...
}

int fun2(const float a){
    // ...
}
```

Putting the *Fun* in *Function* II

Wird dies zu einem Compilerfehler führen?

```
int fun2(const int a){
    // ...
}

int fun2(const float a){
    // ...
}
```

Antwort: Nein, weil die beiden Funktionen unterschiedliche Argumenttypen besitzen (`int` vs `float`)

Putting the *Fun* in *Function* III

Wird dies zu einem Compilerfehler führen?

```
int fun3(const int a){
    // ...
}

int fun3(const int b){
    // ...
}
```

Putting the *Fun* in *Function* III

Wird dies zu einem Compilerfehler führen?

```
int fun3(const int a){
    // ...
}

int fun3(const int b){
    // ...
}
```

Antwort: Ja, weil die beiden Funktionen sich nicht in der Anzahl oder Typ(en) ihrer Argumente unterscheiden.

Putting the *Fun* in *Function* III

Wird dies zu einem Compilerfehler führen?

```
int fun3(const int a){
    // ...
}

int fun3(const int b){
    // ...
}
```

Antwort: Ja, weil die beiden Funktionen sich nicht in der Anzahl oder Typ(en) ihrer Argumente unterscheiden.

Merke: Die Namen der Funktionsparameter sind für den Compiler irrelevant!

Putting the *Fun* in *Function* IV

Wird dies zu einem Compilerfehler führen?

```
int fun4(const int a){
    // ...
}

double fun4(const int a){
    // ...
}
```

Putting the *Fun* in *Function* IV

Wird dies zu einem Compilerfehler führen?

```
int fun4(const int a){
    // ...
}

double fun4(const int a){
    // ...
}
```

Antwort: Ja, weil die beiden Funktionen sich nicht in der Anzahl oder Typ(en) ihrer Argumente unterscheiden.

Putting the *Fun* in *Function* IV

Wird dies zu einem Compilerfehler führen?

```
int fun4(const int a){
    // ...
}

double fun4(const int a){
    // ...
}
```

Antwort: Ja, weil die beiden Funktionen sich nicht in der Anzahl oder Typ(en) ihrer Argumente unterscheiden.

Merke: Die Wiedergabetypen sind für den Compiler irrelevant!

Putting the *Fun* in *Function* V

Wird dies zu einem Compilerfehler führen?

```
int fun5(const int a){
    // ...
}

int fun6(const int a){
    // ...
}
```

Putting the *Fun* in *Function* V

Wird dies zu einem Compilerfehler führen?

```
int fun5(const int a){
    // ...
}

int fun6(const int a){
    // ...
}
```

Antwort: Nein, weil die beiden Funktionen unterschiedlich Namen tragen

Genau mein Typ

```
void out(const int i){
  std::cout << i << " (int)\n";
}
void out(const double i){
  std::cout << i << " (double)\n";
}

int main(){
  out(3.5);
  out(2);
  out(2.0);
  out(0);
  out(0.0);
  return 0;
}
```

Wie wird die Ausgabe aussehen?

Genau mein Typ

```
void out(const int i){
  std::cout << i << " (int)\n";
}
void out(const double i){
  std::cout << i << " (double)\n";
}

int main(){
  out(3.5);
  out(2);
  out(2.0);
  out(0);
  out(0.0);
  return 0;
}
```

Wie wird die Ausgabe aussehen?

- 3.5 (double)
- 2 (int)
- 2 (double)
- 0 (int)
- 0 (double)

Fragen/Unklarheiten?

4. Aufgabe "Tribool"

Tribool als Logik-Objekt

NOT(A)

A	¬A
F	T
U	U
T	F

AND(A,B)

A ∧ B		B		
		F	U	T
A	F	F	F	F
	U	F	U	U
	T	F	U	T

OR(A,B)

A ∨ B		B		
		F	U	T
A	F	F	U	T
	U	U	U	T
	T	T	T	T

F = FALSE, U = UNKNOWN, T = TRUE

- Wie könnten wir das in C++ implementieren?
- Welche Operationen und Werte brauchen wir?

Exercise "Tribool"

```
class Tribool {
private:
    // 0 means false, 1 means unknown, 2 means true.
    unsigned int value;  // INV: value in {0, 1, 2}.
public:
    // ...
};
```

Exercise "Tribool"

```
class Tribool {
private:
    // ...
public:
    // Constructor 1 (passing a numerical value)
    // PRE: value in {0, 1, 2}.
    // POST: tribool false if value was 0, unknown if 1, and true if 2.
    Tribool(unsigned int value_int);
    // TODO: add the definition in tribool.cpp

    // Constructor 2 (passing a string value)
    // PRE: value in {"true", "false", "unknown"}.
    // POST: tribool false, true or unknown according to the input.
    // TODO: add declaration here and the definition in tribool.cpp
    // ...
};
```

Exercise "Tribool"

```
class Tribool {
private:
    // ...
public:
    // ...
    // Member function string()
    // POST: Return the value as string
    // TODO: add declaration here and the definition in tribool.cpp

    // Operator && overloading
    // POST: returns this AND other
    // TODO: add declaration here and the definition in tribool.cpp
};
```

Exercise "Tribool"

Wo fangen wir überhaupt an?

1. Erster (`int`) Constructor
2. Zweiter (`std::string`) Constructor
3. Memberfunktion `string()` implementieren
4. Logisches UND als Operatoren implementieren

Wohin mit all dem?

- Deklarationen ins `Tribool.h`
- Definitionen ins `Tribool.cpp`
  - Mit Out-of-Class-Definitionen mittels Scope Resolution Operator `::`

Let’s Code (gemeinsam!)!

- Öffnet "Tribool" auf **code** expert
- Wir machen eine live Coding-Session

Exercise "Tribool" Konzepte

Folgenden Konzepten und Keywords sind wir beim Lösen dieser Aufgabe begegnet:

- Classes und Structs
- Visibility
- Operator Overloading
- Deklaration vs Definition
- Out-of-Class-Definitionen
- `const` Funktionen
- Konstruktoren ("C-tors")
- Member Initializer Lists
- …

Fragen/Unklarheiten?

5. Iteratoren

Was sind Iteratoren überhaupt?

- Iteratoren werden verwendet, um durch Elemente in einem Container zu iterieren
- Und was sind Container?
  - Container sind Objekte, die zum Speichern von Sammlungen von Elementen verwendet werden
  - Zu den gängigen C++-Containern gehören:
    - `std::vector`
    - `std::set`
    - `std::list`
  - Eine vollständige Liste der Container der C++-Standardbibliothek ist hier zu finden[1]

[1] `https://en.cppreference.com/w/cpp/container`

Iteratoren auf Containern verwenden

Sehr einfach und absichtlich immer gleich!

Gegeben sei ein Container names `C`

- `auto`[2] `it = C.begin()`

[2] Sehr nützlich für unhantliche Wiedergabetypen
[3] PTE: Past-the-End

Iteratoren auf Containern verwenden

Sehr einfach und absichtlich immer gleich!

Gegeben sei ein Container names `C`

- `auto`[2] `it = C.begin()`
  Iterator, der auf das erste Element zeigt
- `auto it = C.end()`
  Iterator, der auf das Element *hinter dem letzten Element* zeigt[3]
- `*it`
  Zugriff (und eventuell Änderung) auf das aktuelle Element
- `++it`
  Iterator um ein Element weitersetzen

[2] Sehr nützlich für unhantliche Wiedergabetypen
[3] PTE: Past-the-End

6. Aufgabe "Find Max"

Aufgabe "Find Max"

```
// PRE:  i < j <= v.size()
// POST: Returns the greatest element of all elements
//       with indices between i and j (excluding j)
unsigned int find_max(const std::vector<unsigned int>& v,
                      unsigned int i,
                      unsigned int j){
  unsigned int max_value = 0;

  for (; i < j; ++i) {
    if (max_value < v.at(i)) {
      max_value = v.at(i);
    }
  }

  return max_value;
}
```

Aufgabe "Find Max"

- Öffnet "Find Max" auf **code** expert
- Überlegt euch, wie ihr das Problem mit Stift und Papier angehen würdet
- Programmiert eine Lösung (optional in Gruppen)

Aufgabe "Find Max" (Lösung)

```
// PRE:  (begin < end) && (begin and end must be valid iterators)
// POST: Return the greatest element in the range [begin, end)
unsigned int find_max(std::vector<unsigned int>::iterator begin,
                      std::vector<unsigned int>::iterator end) {
  unsigned int max_value = 0;

  for(; begin != end; ++begin) {
    if (max_value < *begin) {
      max_value = *begin;
    }
  }

  return max_value;
}
```

Fragen/Unklarheiten?

Die `algorithm` Library

- Sicherlich hat jemand Schlaueres bereits alle gängigen Algorithmen für uns implementiert, oder?
- Ja! Die `algorithm`-Library!
- Diese Funktionen sind so konzipiert, dass sie mit verschiedenen Containern wie Vektoren, Arrays, Listen usw. arbeiten und dabei helfen, Aufgaben effizient auszuführen, ohne dass die Algorihmen jedes mal von Grund auf neu geschrieben werden müssen
- Nicht vergessen: `#include <algorithm>`

Aufgabe "The algorithm Library"

- Öffnet "The algorithm Library" auf **code** expert
- Überlegt euch, wie ihr das Problem angehen würdet
- Programmiert eine Lösung (optional in Gruppen)

Aufgabe "The algorithm Library" (Lösung)

```
// ...

int largest_element = *std::max_element(vec.begin(), vec.end());

// ...

std::sort(vec.begin(), vec.end());

// ...
```

Fragen/Unklarheiten?

7. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche!