Department I - C Plus Plus

Modern and Lucid C++ for Professional Programmers

Week 15 – Exam Preparation

Thomas Corbat / Prof. Peter Sommerlad
Rapperswil, 08.01.2019
HS2018

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS
INSTITUTE FOR
SOFTWARE

- **Durchführung**
  - Mittwoch 30.01.2019 15:10-17:10 Uhr
  - 4.101 (Aula)
- **Rahmenbedingungen**
  - 120 Minuten
  - Fragen sind auf Deutsch
  - Open Book: Papierunterlagen sind erlaubt, ausser alte Prüfungen (CPl/CPlA) oder Abschriften davon
  - Keine elektronischen Hilfsmittel

- **Nahe an den Übungsaufgaben / Testaten. Ja, man muss Code lesen und schreiben!**
  - Vererbung mit Ausgabe (Birbs / Monster)
  - Value-Wrapper (Word-Klasse)
  - Iteratoren-Container-Algorithmen
  - Template Aufgabe mit zu adaptierendem Container (IndexableSet)
  - Constness-Aufgabe
  - Streams and States
  - Value vs. Reference Semantics
- **Multiple-Choice Fragen zu allen Teilen**
- **CUTE Tests muss man lesen und verstehen können**
- **Includes sind wichtig**

Literal Value Examples

Literal Example	Type	Value
'a' '\n' '\x0a'	char char char	Letter a, value: 97 <NL> character, value: 10 <NL> character, value: 10
1 42L 5LL int{} (not really a literal)	int long long long int	1 42 5 0 (default value)
1u 42ul 5ull	unsigned int unsigned long unsigned long long	1 42 5
0X20 0X1F 0XFULL	int int unsigned long long	16 (octal 20) 31 (hex 1F) 15 (hex F)
0.f .33 1e9 42.E-12L .3l	float double double long double long double	0 0.33 1000000000 0.000000000042 (42*10^-12) 0.3
"hello" "\012\n\\"	char const [6] char const [4]	Array of 6 chars: h e l l o <NUL> Array of 4 chars: <NL> <NL> \ <NUL>

Werden alle Potenzen in double gecastet? 1e9

Was bedeutet <NL> beim char?

A Good Class

- **Does one thing well and is named after that**
  - High Cohesion
- **Consists of member functions with only a few lines**
  - Avoid deeply nested control structures
- **Has a class invariant**

  Was bedeutet "invariant"?
  - Provides a guarantee about its state (values of the member variables)
  - Constructors establish that invariant
- **Is easy to use without complicated protocol sequence requirements**

GoodClassName.h

```
class <GoodClassName> {

  <member variables>

  <constructors>

  <member functions>
};
```

Wie funktioniert der Konstruktor mit der Initializer List genau und was ist der Sinn dahinter?

- **Initializer List Constructor**

```
Container box{item1, item2, item3};
```

  - Has one `std::initializer_list` parameter
  - Does not need to be **`explicit`**, implicit conversion is usually desired

```
struct Container {
  Ccontainer() = default;
  Container(std::initializer_list<Element> elements);
private:
  std::vector<Element> elements{};
};
```

- **Initializer List constructors are preferred if a variable is initialized with {}**

```
std::vector v(5, 10)
```

```
std::vector v{5, 10}
```

- **Keywords for defining a class**
  - `class`
  - `struct`
- **Default visibility for members of the class are**
  - `private` for `class`
  - `public` for `struct`

```
struct <name> {
  ...
};
```

Default Vererbung bei Structs und Klassen wenn kein Keyword angegeben wird
o Wie ist die Vererbung z.B `class Base : Sub`
o Wie ist die Vererbung z.B `struct Base : Sub`

Date.h

```
class Date {
  int year, month, day;
public:
  Date(int year, int month, int day)
  : year{year}, month{month}, day{day} {/*...*/}

  static bool isLeapYear(int year) {/*...*/}

private:
  bool isValidDate() const {/*...*/}
};

#endif /* DATE_H_ */
```

- **Base classes are specified after the name**

```
class <name> : <base1>, ..., <baseN>
```

- **Multiple inheritance is possible**
- **Inheritance can specify a visibility**
  - `public, protected, private`
  - Limits the **maximum** visibility of the inherited members
- **Details about Diamonds and Virtual inheritance later**

```
class Base {
private:
  int onlyInBase;
protected:
  int baseAndInSubclasses;
public:
  int everyoneCanFiddleWithMe
};
```

```
class Sub : public Base {
  //Can see baseAndInSubclasses and
  //everyoneCanFiddleWithMe
};
```

Right or Wrong?

?

```
struct Recipe {
  Recipe(std::vector<Step> steps) = default;
  Meal cook() const;
private:
  std::vector<Step> steps{};
```

Incorrect

Of all constructors, only default, copy and move constructors can be declared = `default`.

Was bedeutet das explicit Kennwort und wann wird es genau angewendet?

```
struct Chair {
  explicit Chair(unsigned legs = 4u);
private:
  unsigned legs;
};
Chair::Chair(unsigned legs)
  : legs{legs} {
  if (legs < 1u) {
    throw std::invalid_argument{"..."};
  }
}
```

Correct

Constructors can have default arguments too. In the declaration, a constructor that can be called with a single argument should be explicit. It uses the member initializer list and throws an exception if the invariant cannot be established.

- **Typeconversion Constructor**

```
Date tomorrow{"19/10/2017"s};
```

  - Has one `<other-type> const &` parameter
  - Converts the input type if possible
  - Declare **`explicit`** to avoid unexpected conversions!

```
class Date {
public:
  Date(int year, int month, int day);
  //Default-Constructor
  Date();
  //Copy-Constructor
  Date(Date const &);
  //Move-Constructor
  Date(Date &&);
  //Typeconversion-Constructor
  explicit Date(std::string const &);
  //Destructor
  ~Date();
};
```

Breaking Cycles Using `std::weak_ptr`

- **The `std::shared_ptr` cycles need to be broken**

```
struct Person {
  std::shared_ptr<Person> child;
  std::weak_ptr<Person> parent;
};

int main() {
  auto anakin = std::make_shared<Person>();
  auto luke = std::make_shared<Person>();
  anakin->child = luke;
  luke->parent = anakin;
  //...
}
```

Was passiert, wenn ich das Child(Luke) lösche, mit dem Shared Pointer von Anakin?