Überladene Funktionen einfacher auflösen

Mehr Spaß mit Templates …

Stefan Pabst

INFORM

Das Problem: Überladene Funktionen übergeben

2015-01-08

Vortrag „Qt Signal und C++11 Lambdas” von Sven Johannsen:

- ```
  connect(spin,
  static_cast<void(QSpinBox::*)(int)>(&QSpinBox::valueChanged),
    label, static_cast<void(QLabel::*)(int)>(&QLabel::setNum));
  ```
  ```

- `int f(int);`
- `string f(string);`

- `void g1(int(*f)(int));`
- ```
  template <typename R>
  void g2(R(*f)(int));
  ```
- ```
  template <typename R, typename... Args>
  void g3(R(*f)(Args...));
  ```
- ```
  //template <typename... Args, typename R>
  //void g4(R(*f)(Args...));
  ```

- `g1(f);`
- `g2(f);`
- `//g3(f);`
- `g3<int, int>(f);`
  ```

Mögliche Lösung: Eigener Typ für Argumentliste

- `int f(int);`
- `string f(string);`
-
```
template <typename... Args>
class Arg_List{};
```
- `g<Arg_List<int>>(f);`
-
```
template <typename T>
class H;
```
-
```
template <typename... Args>
class H<Arg_List<Args...>>
public:
  template <typename R>
  using Type = R(*)(Args...);
};
```
-
```
template <typename T, typename R>
void g(H<T>::Type<R> f);
```

Mögliche Lösung: Eigener Typ für Argumentliste

- `int f(int);`
- `string f(string);`
-
```
template <typename... Args>
class A{};
```
- `g<A<int>>(f);`
-
```
template <typename T>
class H;
```
-
```
template <typename... Args>
class H<A<Args...>>
public:
  template <typename R>
  using Type = R(*)(Args...);
};
```
-
```
template <typename T, typename R>
void g(H<T>::Type<R> f);
```

Meine Lösung: Parameter aufteilen auf Klasse und Funktion

- `int f(int);`
- `string f(string);`

- `C<int>::g(f);`

```
template <typename... Args>
class C{
public:
  template <typename R>
  static void g(R(*f) (Args...));
};
```

Meine Lösung: Parameter aufteilen auf Klasse und Funktion

- `int f(int);`
- `string f(string);`

- `C<int>()(f);`

```
template <typename... Args>
class C{
public:
  template <typename R>
  void operator()(R(*f) (Args...));
};
```

Feinschliff

```
template <typename... Args>
class C{
public:
  template <typename R>
  void operator()(R(*f) (Args...));
};
```

Feinschliff – „using“ für Funktionstyp

```
template <typename... Args>
class C{
public:
  template <typename R>
  using Fkt = R(*)(Args...);

  template <typename R>
  void operator()(Fkt<R> f);
};
```

Feinschliff – Rückgabetyp & Implementierung

```
template <typename... Args>
class C{
public:
  template <typename R>
  using Fkt = R(*)(Args...);

  template <typename R>
  Fkt<R> operator()(Fkt<R> f){
    return f;
  }
};
```

Feinschliff – Memberfunktionen

- ```
  template <typename... Args>
  class C{
  public:

    […]

    template <typename R, typename T>
    using Fkt_m = R (T::*) (Args...);

    template <typename R, typename T>
    Fkt_m<R, T> operator()(Fkt_m<R, T> f){
      return f;
    }
  };
  ```
- Anmerkung: Kompiliert noch nicht in Visual Studio (as of VS2013) [EDIT: https://connect.microsoft.com/VisualStudio/feedback/details/800231/c-11-alias-template-issue]
  ```

Feinschliff – konstante Funktionen

- […]

```
template <typename R>
using Fkt_c = R (*) (Args...) const;

template <typename R>
Fkt_c<R> operator()(Fkt_c<R> f){
  return f;
}

template <typename R, typename T>
using Fkt_m_c = R(T::*) (Args...) const;

template <typename R, typename T>
Fkt_m_c<R, T> operator()(Fkt_m_c<R, T> f){
  return f;
}
```

[…]

TODO: ~~`static`~~
[EDIT: `volatile`]

Left as exercise …

Die Lösung: Überladene Funktionen auflösen

2015-01-08

Vortrag „Qt Signal und C++11 Lambdas” von Sven Johannsen:

```
connect(
  spin, static_cast<void(QSpinBox::*)(int)>(&QSpinBox::valueChanged),
  label, static_cast<void(QLabel::*)(int)>(&QLabel::setNum));
```

```
connect(
  spin, C<int>()(&QSpinBox::valueChanged),
  label, C<int>()(&QLabel::setNum));
```

Nachtrag

- Feuer gefangen?
- Verbesserungen?
- Alternative Lösungen?
- Ideen, Anmerkungen, Fragen …?

- ➔ Stefan.Pabst@rwth-aachen.de