Einführung in die Programmierung mit C++

Modern Family: Newton und Strukturierung des Quellcodes

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Inhalt

Eine Datei

Templatebibliothek

- Verzeichnisstruktur
- Implementierung
- Verwendung
- Übersetzung mit `make`

Inhalt

Eine Datei

Siehe MF_Newton.cpp für

- Verwendung von std::vector
- Parameterübergabe
- Typgenerik

→ Live

Inhalt

- Ziel:
  - Trennung der Templates für Modell (model), Fehlerquadrate und deren Ableitungen (error, d_error, dd_error), Parameterschätzer (estimate) und Ein-/Ausgabe (read_data, write_data) vom Quellcode des Hauptprogramms (main)
  - Trennung der Templates für (Vorwärts-)Deklaration und Definition (Implementierung) ermöglicht Verwendung innerhalb mehrerer Übersetzungseinheiten des Anwendungsquellcodes
- Lösung: Templatebibliotheken libmodel, liberror, libestimate, libio
- Abhängigkeiten:
  - libmodel $\rightarrow^{1}$ liberror
  - liberror $\rightarrow$ libestimate
  - libestimate und libio $\rightarrow$ main

[1] ... wird benötigt durch ...

Templatebibliothek

Verzeichnisstruktur


```
liberror // error, d_error, dd_error
    error.hpp
    error.cpp
libestimate // estimate
    estimate.hpp
    estimate.cpp
libio // read_data, write_data
    io.hpp
    io.cpp
libmodel // model
    model.hpp
    model.cpp
main // main
    data.txt
    Makefile
    MF_Newton.cpp
```

Beispiel: libmodel

- Deklaration: libmodel/model.hpp

```
1 template<typename T>
2 T model(T, T);
```

- Definition: libmodel/model.cpp

```
1 #include <cmath>
2
3 template<typename T>
4 T model(T p, T x) { return pow(p*x,2); }
```

Beispiel: liberror in libestimate

```
1  #include "error.hpp" // (forward) declares d_error, dd_error
2
3  #include <cmath>
4  #include <vector>
5
6  template<typename T>
7  T estimate(T p, const std::vector<T> &x, const std::vector<T> &y, T eps) {
8    while (fabs(d_error(p,x,y))>eps)
9      p-=d_error(p,x,y)/dd_error(p,x,y);
10   return p;
11 }
12
13 #include "error.cpp" // defines / implements d_error, dd_error
```

Templatebibliothek

Verwendung in Anwendungsquellcode


```
1  #include "io.hpp"
2  #include "estimate.hpp"
3  ...
4  int main(int argc, char *argv[]) {
5    ...
6    read_data(in,p,x,y);
7    ...
8    p=estimate(p,x,y,sqrt(std::numeric_limits<T>::epsilon()));
9    ...
10   write_data(out,p,x,y);
11   ...
12   return 0;
13 }
14
15 #include "io.cpp"
16 #include "estimate.cpp"
```

Übersetzung mit make

```
1  EXE=$(addsuffix .exe, $(basename $(wildcard *.cpp)))
2  CPPC=g++ -std=c++17
3  CPPC_FLAGS=-Wall -Wextra -pedantic -g
4  LIBIO_DIR=../libio
5  LIBMODEL_DIR=../libmodel
6  LIBERROR_DIR=../liberror
7  LIBESTIMATE_DIR=../libestimate
8  INC_DIRS=-I$(LIBIO_DIR) -I$(LIBMODEL_DIR) -I$(LIBERROR_DIR) -I$(
       LIBESTIMATE_DIR)
9
10 all : $(EXE)
11
12 %.exe : %.cpp
13         $(CPPC) $(CPPC_FLAGS) $(INC_DIRS) $< -o $@
14
15 clean :
16         rm -f $(EXE)
17
18 .PHONY: all clean
```

Zusammenfassung

Eine Datei

Templatebibliothek

- Verzeichnisstruktur
- Implementierung
- Verwendung
- Übersetzung mit `make`