Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

Einführung in die Programmierung mit C++

Übungsblatt 2

Einlesen von Dateien
Graphen

Sebastian Christodoulou Alexander Fleming Uwe Naumann

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

RWTH Aachen

1. Die abgegebene .cpp Datei:

```
// Der Preprocessor hängt den Inhalt von iostream hier an
#include <iostream>

// main: Diese Funktion wird immer beim ausführen der ausführbaren Datei aufgerufen
int main()
{
  // Der Text in Anführungszeichen wird zu std::cout gestreamt. D.h., er erscheint
  // im Terminal. eine Leerzeile wird mittels std::endl angefügt
  std::cout << "Willkommen zur Einfuehrung in die Programmierung!" << std::endl;

  // Gibt 0 and das Betriebssystem zurück ("Error-code" für erfolgreiches Ausführen)
  return 0;
}
```

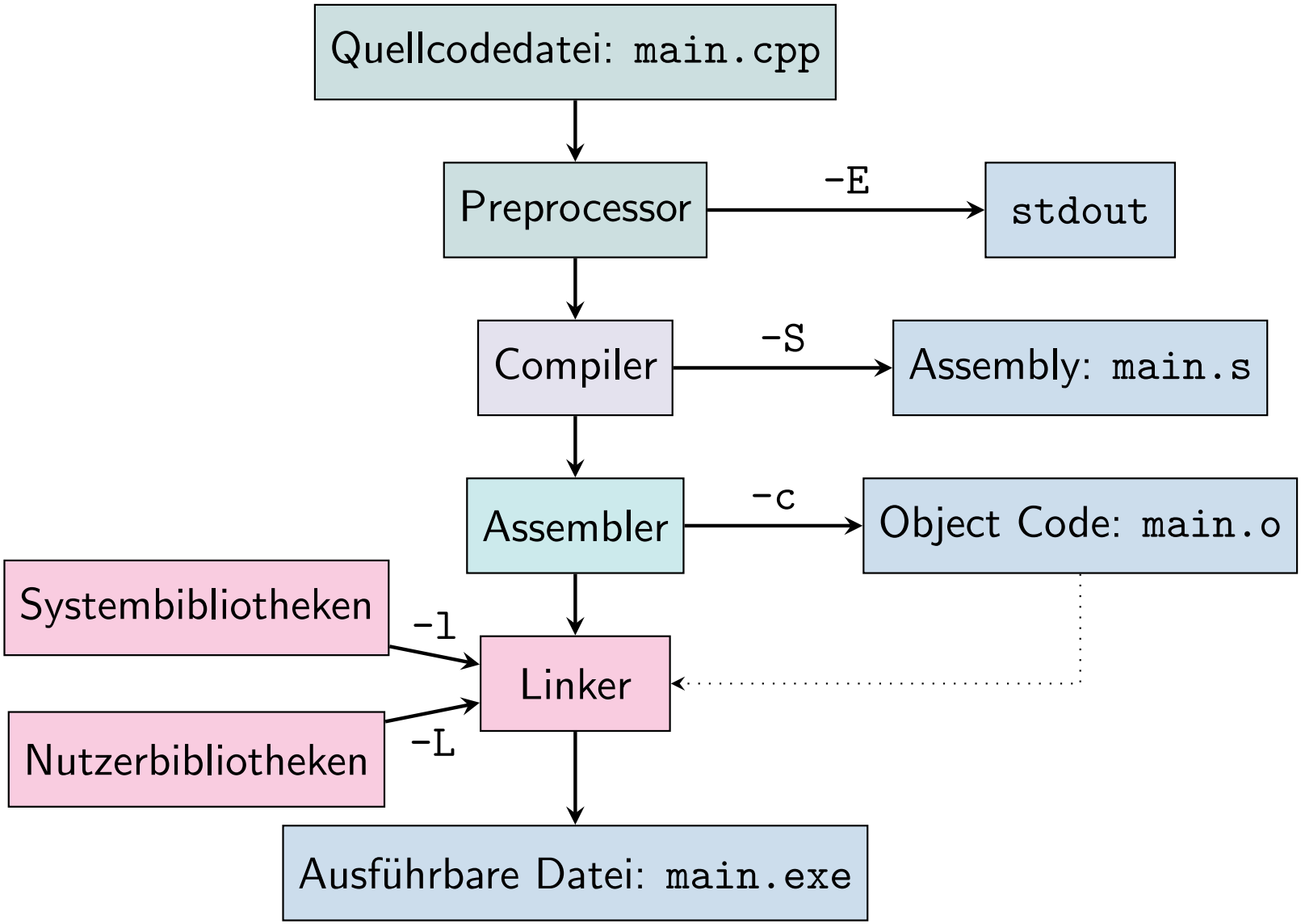
2. Versehen Sie die einzelnen Zeilen mit erklärenden Kommentaren.

3. Erläutern Sie kurz, was die folgenden Compileraufrufe bewirken.

```
g++ -E hello.cpp
g++ -c hello.cpp
g++ hello.o
g++ -o hello hello.cpp
g++ -Wall -Werror hello.cpp
```

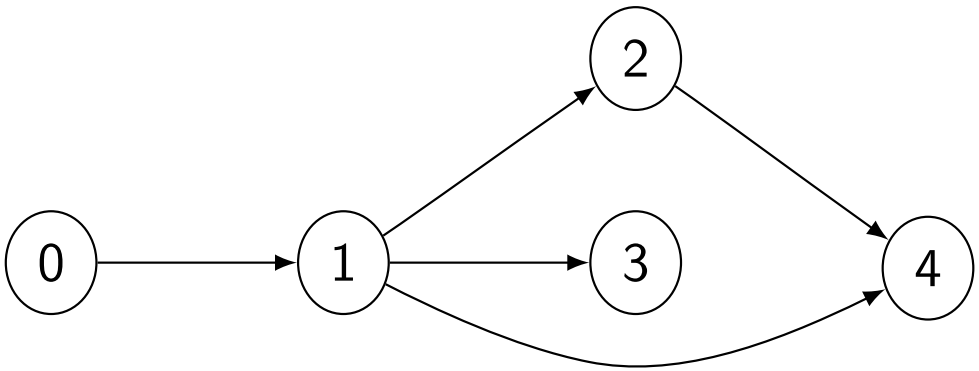
- `g++ -E hello.cpp` führt den Preprocessor aus und leitet die Ergebnisse zu `stdout`
- `g++ -c hello.cpp` führt den Preprocessor, Compiler, und Assembler auf `hello.cpp` aus und produziert die Objektdatei `hello.o`
- `g++ hello.o` linkt `hello` (mit den Systembibliotheken). Dies ergibt die ausführbare Datei `a.out`
- `g++ -o hello hello.cpp` führt den Preprocessor, Compiler, Assembler, und Linker auf `hello.cpp` aus ergibt die ausführbare Datei `hello`
- `g++ -Wall -Werror hello.cpp` Führt den Kompilierungsprozess aus, aber *alle* Warnungen werden angezeigt (`-Wall`), und alle Warnung werden als Kompilierungsfehler betrachtet (`-Werror`).

Phasen von g++


Quellcodedatei: main.cpp
Preprocessor
-E
stdout
Compiler
-S
Assembly: main.s
Assembler
-c
Object Code: main.o
Systembibliotheken
-l
Linker
Nutzerbibliotheken
-L
Ausführbare Datei: main.exe

Ein Graph G besteht aus Knoten ($\circ$) und Kanten ($\rightarrow$). Knoten sind durch Zahlen identifiziert. Der untere Graph hat 5 Knoten $[0, \ldots, 4]$.

```
5
0 1
1 2
1 3
2 4
1 4
```



Kanten beschreiben, welcher Knoten über welchen anderen erreichbar ist. Die Datei `my_graph.txt` beschreibt den rechten Graph mit 5 Kanten durch Anfangs- und Endknoten:

- von 0 erreicht man 1
- von 1 erreicht man 2
- von 1 erreicht man 3
- von 2 erreicht man 4
- von 1 erreicht man 4

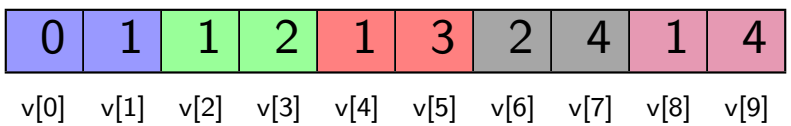
Diese Textdatei werden wir auslesen, und dann visualisieren.

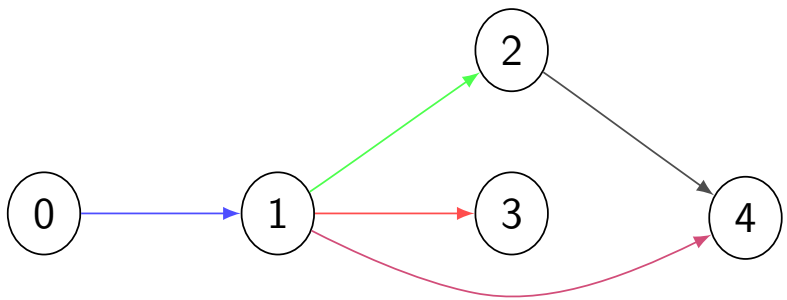
In der Datei vom obigen Speicherformat

- gibt die *erste* Zahl an, wie viele Kanten ausgelesen werden müssen.
- Die folgenden *Zahlenpaare* beschreiben jeweils eine Kante.

Wir kodieren diese Kanten in einen `std::vector<int>` wie folgt:

- Jeder *gerade* Eintrag des Vektors beschreibt den Anfangspunkt einer Kante
- Jeder darauffolgende *ungerade* Eintrag beschreibt den Endpunkt derselben Kante.





... somit können wir mittels `std::vector<int>` (links) einen Graph (rechts) repräsentieren.

1. Öffnen Sie `my_graph.txt` mittels `std::ifstream`, und lesen Sie dessen Kanten wie beschrieben in einen `std::vector<int> v` passender Größe aus.
2. Geben Sie die Einträge von `v` auf der Kommandozeile durch einen `for`-loop aus.
3. Initialisieren Sie nun einen zweiten Vektor `v2` mit 2 mehr Einträgen als `v`. Kopieren Sie alle Einträge von `v` in `v2`.
4. Zwei Einträge in `v2` sind nun 'leer'. Setzen Sie diese so, dass diese eine weitere Kante $3 \rightarrow 4$ repräsentieren.
5. Rufen Sie die Funktion `graph_to_dot` auf `v2` auf. Diese erstellt eine Datei `graph.dot` in Ihrem momentanen Verzeichnis.
6. Sie können den entstandenen `graph.dot` mittels Graphviz visualisieren. Tun Sie dies indem Sie den Befehl
   `$ dot -Tpdf graph.dot > graph.pdf`
   in der Kommandozeile ihres Betriebssystems ausführen
7. Öffnen Sie `graph.pdf`. Vergleichen Sie das Ergebnis mit dem Graph auf Seite 5. Wurde die richtige Kante hinzugefügt?

Abgaben

- main.cpp
- graph.pdf