

C/C++ Materialpaket (Level C)

02d_FLOW_adv – Control Flow (advanced)

Prof. Dr. Carsten Link

Zusammenfassung

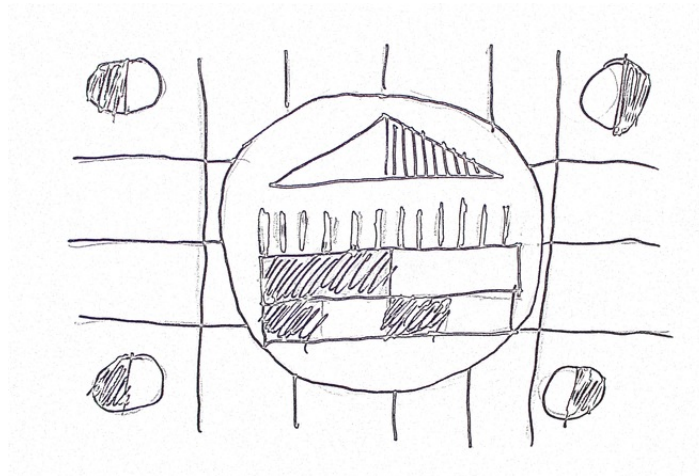


Abbildung 1: Testbild

Inhaltsverzeichnis

1	Kompetenzen und Lernergebnisse	2
2	Konzepte	2
2.1	Grundlegender Kontrollfluss	2
2.1.1	Futures und Threads	2
2.1.2	Exception	5
2.1.3	Lambda Expression	6
3	Material zum aktiven Lernen	7
3.1	Aufgabe: Grundgerüst	7
3.2	Aufgabe: Modifikationen	7
3.3	Verständnisfragen	7
4	Literatur	8

1 Kompetenzen und Lernegebnisse

Durch das Bearbeiten dieses Materialpaketes erwerben Sie diese Kompetenzen (Wissen, Fähigkeiten und Fertigkeiten zur Problemlösung):

Sie können vorhersagen, wie sich die Ausführung von Anweisungen aufgrund der komplexeren Kontrollstrukturen zur Laufzeit verhalten wird und umgekehrt zu einer geforderten Ablaufreihenfolge Quelltext mit den entsprechenden Kontrollstrukturen konstruieren.

Die oben genannten Kompetenzen erwerben Sie, indem Sie Lernziele erreichen, welche sich prüfen lassen. Lernegebnisse: Sie können nachweislich¹:

- Code erstellen, dessen Zeilen/Anweisungen in einer vorgegebenen Reihenfolge abgearbeitet werden
- Code analysieren, um die Reihenfolge der Abarbeitung der darin enthaltenen Zeilen/Anweisungen zu ermitteln
- die Unterschiede und Gemeinsamkeiten von TBD benennen und in Implementierungen ausnutzen

2 Konzepte

Im Folgenden wird auf die grundlegenden Kontrollstrukturen von C++ eingegangen. Diese Kontrollstrukturen erlauben es dem Programmierer, zu steuern, in welcher Abfolge einzelne Anweisungen und Ausdrücke zu Laufzeit des Programms ausgeführt bzw. ausgewertet werden.

2.1 Grundlegender Kontrollfluss

Im Folgenden wird ein Makro verwendet, der hilft, den Kontrollfluss in einem Programm sichtbar zu machen. Die Funktionen in `printSteps.cpp` sorgen dafür, dass der Quelltext auf der Konsole ausgegeben wird. Der in `printSteps.hpp` definierte Makro `nop` sorgt anschließend dafür, dass eine Schrittnummer über den Quelltext geschrieben wird, sobald der Kontrollfluss über diesen Makro wandert.

Es brauchen also nur die Schrittnummern in aufsteigender Reihe verfolgt zu werden, um den genauen Ablauf eines Programms erkennen zu können.

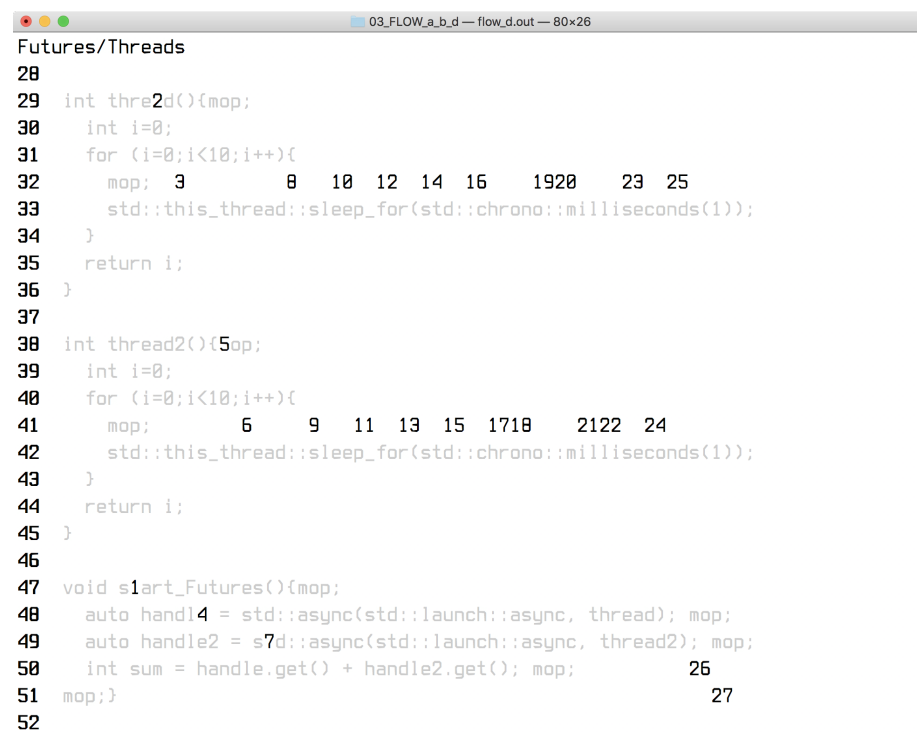
2.1.1 Futures und Threads

```
25 #define mop m.lock();nop;m.unlock();
26
27 std::mutex m;
28
```

¹Sie können das Erzielen der einzelnen Lernergebnisse beispielsweise bei einem Testat im Praktikum oder einer Aufgabe in der Modulprüfung nachweisen.

```
29 int thread(){mop;
30     int i=0;
31     for (i=0;i<10;i++){
32         mop;
33         std::this_thread::sleep_for(std::chrono::milliseconds(1));
34     }
35     return i;
36 }
37
38 int thread2(){mop;
39     int i=0;
40     for (i=0;i<10;i++){
41         mop;
42         std::this_thread::sleep_for(std::chrono::milliseconds(1));
43     }
44     return i;
45 }
46
47 void start_Futures(){mop;
48     auto handle = std::async(std::launch::async, thread); mop;
49     auto handle2 = std::async(std::launch::async, thread2); mop;
50     int sum = handle.get() + handle2.get(); mop;
51 mop;}
```

Wird nun die Funktion `start_Futures()` aufgerufen und ausgeführt, so ergibt sich die in Abbildung 2 dargestellte Ausgabe.



```

Futures/Threads
28
29 int thre2d(){mop;
30     int i=0;
31     for (i=0;i<10;i++){
32         mop; 3      8  10 12 14 16    1920    23 25
33         std::this_thread::sleep_for(std::chrono::milliseconds(1));
34     }
35     return i;
36 }
37
38 int thread2(){5op;
39     int i=0;
40     for (i=0;i<10;i++){
41         mop;      6   9  11 13 15 1718    2122 24
42         std::this_thread::sleep_for(std::chrono::milliseconds(1));
43     }
44     return i;
45 }
46
47 void s1art_Futures(){mop;
48     auto hand14 = std::async(std::launch::async, thread); mop;
49     auto handle2 = s7d::async(std::launch::async, thread2); mop;
50     int sum = handle.get() + handle2.get(); mop;          26
51     mop;}                                                  27
52

```

Abbildung 2: Futures bzw. Threads

2.1.2 Exception

```
54 void thrower(){nop;
55     nop;
56     nop; throw std::string("Exception!"); nop;
57     nop;
58     nop;}
59
60 void start_Exception(){nop;
61     try{nop;
62         nop; thrower(); nop;
63         nop; }catch(std::string){nop;
64             nop;
65             nop; }
66     nop;}
```

Wird nun die Funktion `start_Exception()` aufgerufen und ausgeführt, so ergibt sich die in Abbildung 3 dargestellte Ausgabe.

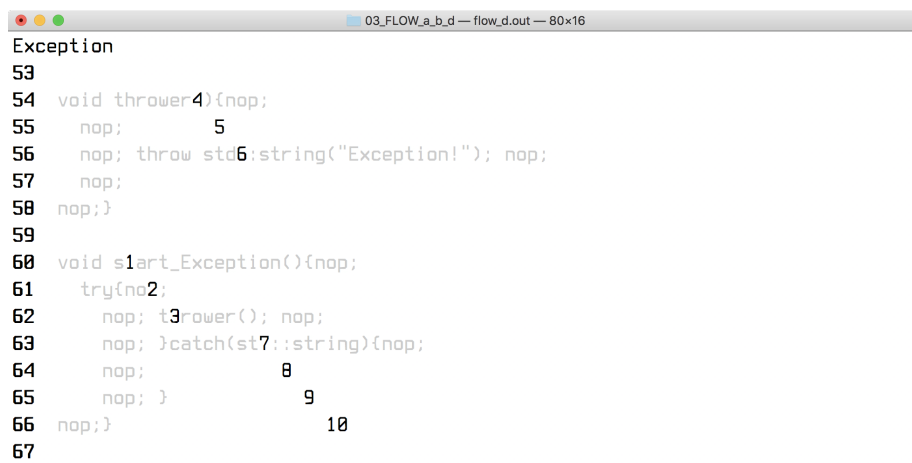


Abbildung 3: Exception

2.1.3 Lambda Expression

Funktionen erlauben es, Werte (die Parameter) an einen Anweisungsblock (den Funktionsrumpf) zu übergeben. Bei Funktionsaufrufen bringen wir Daten zum Code. Lambda Expression erlauben es, einen Anweisungsblock zu übergeben (an eine Funktion oder eine Variable zur späteren Verwendung). Bei Lambda Expressions bringen wir Code zu den Daten. Bei Lambda Expressions ist es besonders nützlich, dass der Anweisungsblock auf Variablen im ihm umgeben Scope verweisen kann.

```

70 void lambdaCaller(std::function<void()> lambda){nop;
71     nop; lambda(); nop;
72 nop;}
73
74 void start_Lambda(){nop;
75     nop;
76     nop; lambdaCaller([](){nop;}); nop;
77     nop;
78 }
```

Wird nun die Funktion `start_Lambda()` aufgerufen und ausgeführt, so ergibt sich die in Abbildung 4 dargestellte Ausgabe.

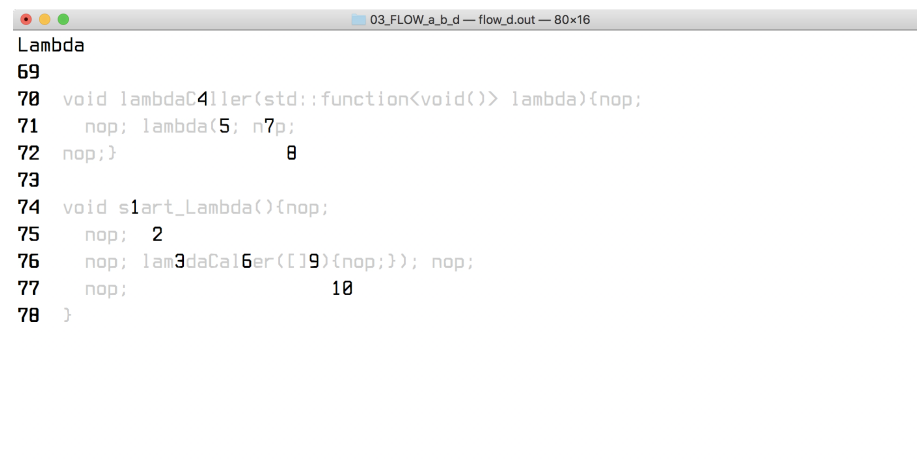


Abbildung 4: Lambda Expression

3 Material zum aktiven Lernen

Regelmäßiger Hinweis: Da eine Programmiersprache nur durch aktive Verwendung erlernt werden kann, werden im Folgenden Aufgaben zum praktischen Üben vorgestellt. Zunächst wird ein Grundgerüst (C/C++-Programm) erstellt, welches dann auf mehrere Arten modifiziert wird. Insbesondere die Modifikationen ermöglichen es dem Lernenden (und auch dem Lehrenden), die Qualität des Kompetenzerwerbs bzgl. dieses Materialpakets bewerten zu können.

3.1 Aufgabe: Grundgerüst

TBD

3.2 Aufgabe: Modifikationen

Regelmäßiger Hinweis: Weiter unten ist eine Liste mit Modifikationen gegeben, die zwei Zwecken dienen: 1) Sie dienen als Richtschnur für das Praktizieren und Üben der Inhalte dieses Materialpakets. 2) Die Modifikationen können im Rahmen eines Testats als Aufgabe verwendet werden, durch deren Lösung Studierende nachweisen können, dass sie den Stoff dieses Materialpakets beherrschen. Stellen Sie sicher, dass Sie jede einzelne der nachfolgenden Modifikationen innerhalb weniger Minuten (ca. 5 - 10) vor Zuschauern (Testatsituation) umsetzen können. Konkret sollen Sie im Testat in der Lage sein, das gegebene Grundgerüst um mindestens eine zufällig ausgewählte Modifikation zu erweitern. Bereiten Sie dazu auf Ihrer Arbeitsumgebung ein Verzeichnis vor, welches ausschließlich das Grundgerüst enthält. **Arbeiten Sie also auf einer Kopie des Verzeichnisses `src-cpp-student/`!** Achten Sie darauf, dass der Text auf Ihrem Bildschirm in heller Umgebung aus einem Meter Abstand heraus gut lesbar ist (light mode, große Schrift).

Modifikationen:

1. TBD
2. Lassen Sie mit dem `PRINT`-Makro folgendes Muster erzeugen:

```
15
```

3.3 Verständnisfragen

Nach Bearbeitung des Kapitels “Konzepte”, der Erstellung des Grundgerüsts sowie dem Üben der Modifikationen sollten Sie in der Lage sein, die folgenden Fragen zu beantworten.

1. Freiwillig: Zeigen Sie auf, inwiefern das Piktogramm auf der Titelseite dieses Materialpaketes den Inhalt zusammengefasst darstellt.
2. TBD

4 Literatur

- [PPP] Stroustrup, Bjarne: Programming - Principles and Practice using C++
- [TCPL] Stroustrup, Bjarne: The C++ Programming Language, Fourth Edition