

C/C++ Materialpaket (Level C)

02a_FLOW_basic – Control Flow (basic)

Prof. Dr. Carsten Link

Zusammenfassung

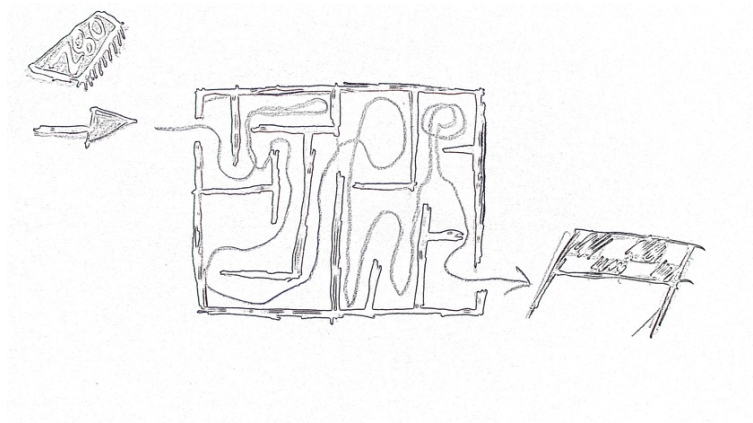


Abbildung 1: Programmfluss – oder wie die CPU durch das Programm läuft

Inhaltsverzeichnis

1	Kompetenzen und Lernegebnisse	2
2	Konzepte	2
2.1	Grundlegender Kontrollfluss	3
2.1.1	Sequenz	3
2.1.2	Selektion	5
2.1.3	Iteration	6
2.1.4	Subroutinen (Funktionen)	7
2.1.5	Rekursion	8
2.2	Micro Style Guide	9
2.2.1	(Level C) Goto, considered harmful	10
3	Material zum aktiven Lernen	11
3.1	Aufgabe: Grundgerüst	11
3.2	Aufgabe: Modifikationen	11
3.3	Verständnisfragen	14

4 Nützliche Links	15
5 Literatur	15

1 Kompetenzen und Lernergebnisse

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 einfachen 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. Lernergebnisse: Sie können nachweislich¹:

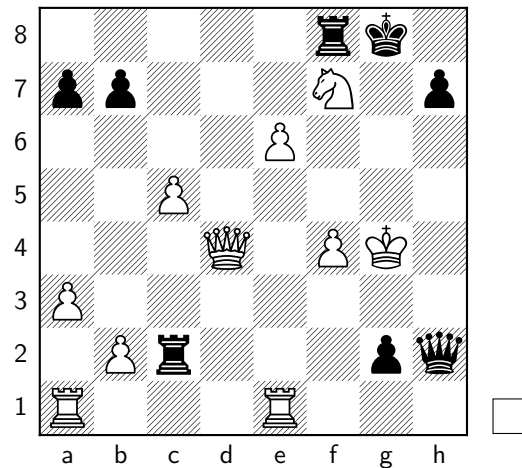
- Code erstellen, dessen Zeilen/Anweisungen in einer vorgegebenen Reihenfolge abgearbeitet werden
- Code analysieren, um die Reihenfolge der Abarbeitung der darin enthaltenen Zeilen/Anweisungen zu ermitteln
- die Abarbeitung von Code in Gedanken simulieren und so den Ablauf eines Programms (Ausführungsschritte, Variablen) voraussagen
- die Unterschiede und Gemeinsamkeiten von Iteration und Rekursion 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.

Ziel ist es hier also, Abläufe im Kopf durchspielen zu können. Ein gutes Beispiel für dieses Durchspielen findet sich beim Schachspiel: hier spielen wir Züge im Kopf durch, um die beste Zugfolge zu finden.

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



In der oben abgebildeten Schachsituation gibt es verschiedene Zugmöglichkeiten. Bei C++-Programmen ist es ähnlich: der konkrete Programmablauf hängt von Werten in Variablen ab, welche zum Teil aus Eingaben zur Laufzeit (Tastatur, Datei, Netzwerk) stammen.

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.

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

2.1.1 Sequenz

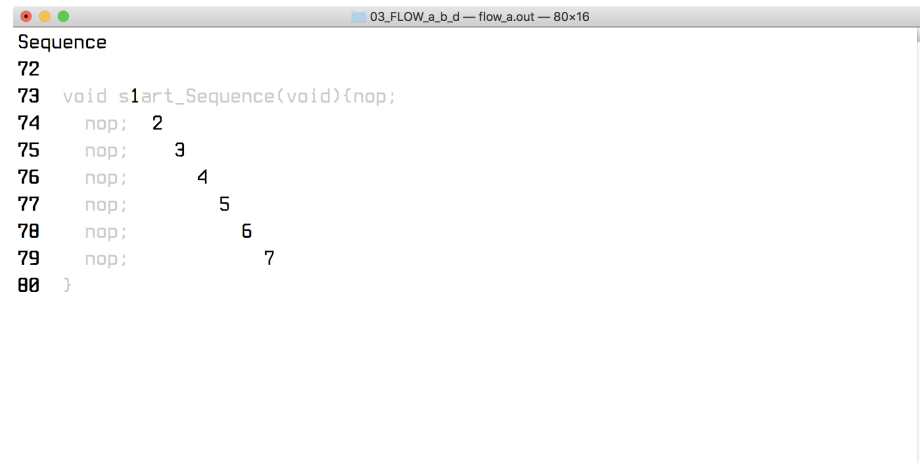
Die Sequenz bildet eine Folge von Anweisungen ab.

```

73 void start_Sequence(void){nop;
74     nop;
75     nop;
76     nop;
77     nop;
78     nop;
79     nop;
80 }
```

Wird nun die Funktion `start_Sequence()` aufgerufen und ausgeführt, so ergibt

sich die in Abbildung 2 dargestellte Ausgabe.



```
Sequence
72
73 void start_Sequence(void){nop;
74     nop; 2
75     nop; 3
76     nop; 4
77     nop; 5
78     nop; 6
79     nop; 7
80 }
```

Abbildung 2: Sequenz

2.1.2 Selektion

Mit der Selektion lässt sich abhängig von einer Bedingung eine von zwei Sequenzen auswählen bzw. überspringen.

```

82 void start_Selection(void){nop;
83     nop; if (13 == 17){
84         nop;
85         nop;
86         nop;
87     }else{
88         nop;
89         nop;
90         nop;
91     }nop;
92 nop;}

```

Wird nun die Funktion `start_Selection()` aufgerufen und ausgeführt, so ergibt sich die in Abbildung 3 dargestellte Ausgabe. Dort wird ersichtlich, dass die Zeilen im `else`-Zweig ausgeführt wurden (da die dortige Bedingung erfüllt war), die des `if`-Zweiges jedoch übersprungen wurden (die umgekehrte Bedingung war ja nicht erfüllt).

```

03_FLOW_a_b_d — flow_a.out — 80x16
Selection
81
82 void start_Selection(void){nop;
83     nop; if (13 == 17){
84         nop;
85         nop;
86         nop;
87     }else{
88         nop;3
89         nop; 4
90         nop; 5
91     }nop; 6
92 nop;} 7
93

```

Abbildung 3: Selektion

2.1.3 Iteration

Die Iteration wiederholt eine Sequenz, solange eine Bedingung erfüllt ist.

```

59 void start_Iteration_all(void){nop;
60   nop; int i=0; const int loops=6;
61   while(i<loops){
62     nop; i++;
63   }
64   for(int k=0;k<loops;k++){
65     nop;
66   }
67   do {
68     nop;
69   } while (--i); nop;
70 nop;}

```

Wird nun die Funktion `start_Iteration_all()` aufgerufen und ausgeführt, so ergibt sich die in Abbildung 4 dargestellte Ausgabe. In der Ausgabe ist zu erkennen, dass die Zeilen im Rumpf der jeweiligen Schleifen mehrfach ausgeführt wird. Die Bedingung für die Wiederholung ist 6 mal erfüllt.

```

Iteration_all
58
59 void start_Iteration_all(void){nop;
60 nop; int i=0; const int loops=6;
61 while(i<loops){
62 nop; i3+4 5 6 7 8
63 }
64 for(int k=0;k<loops;k++){
65 nop; 9 1011121314
66 }
67 do {
68 nop; 151617181920
69 } while (--i); nop; 21
70 nop;} 22
71

```

Abbildung 4: Iteration

2.1.4 Subroutinen (Funktionen)

In C/C++ werden Subroutinen *Funktionen* genannt – unabhängig davon, ob sie einen Wert zurückliefern, oder nicht (auch unabhängig davon, ob sie Seiteneffekte haben oder nicht). Subroutinen erlauben Verzweigung mit Wiederkehr. Außerdem erlauben Subroutinen es, eine Sequenz mit einem Namen zu versehen (dem Funktionsnamen) und an verschiedenen Stellen des Programms mehrfach zu verwenden. Haben Funktionen einen Rückgabewert, so wird nach Ausführung der Funktion diese durch den Rückgabewert ersetzt (im Beispiel unten `int x = func_1(3);` wird `func_1(3)` durch 9 ersetzt, womit x initialisiert wird).

```
96 int func_1(int arg);
97
98 void start_subroutine(void){nop;
99     nop; int x = func_1(3); nop;
100     nop;
101 nop;}
102
103 int func_1(int arg){nop;
104     nop; int local_1 = arg;
105     nop;
106     nop;
107     nop;
108     nop; return arg * local_1;
109 }
```

Wird nun die Funktion `start_subroutine()` aufgerufen und ausgeführt, so ergibt sich die in Abbildung 5 dargestellte Ausgabe. Es ist bei einem Funktionsaufruf eine Ähnlichkeit zur Selektion zu sehen: es wird verzweigt. Der wesentliche Unterschied zeigt sich am Ende des Codes, zu dem verzweigt wurde: es wird danach zu der ursprünglichen Stelle der Verzweigung zurückgekehrt.

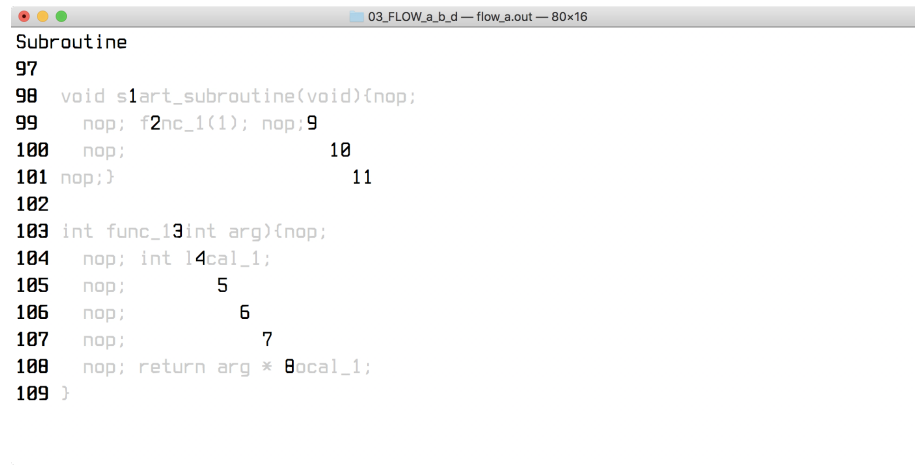


Abbildung 5: Subroutine

2.1.5 Rekursion

```

42 void recurse(int turns){nop;
43   colorOffset++; nop;
44   nop; if(turns>0) { nop;
45     nop;
46     nop; recurse(turns - 1); nop;
47     nop;}
48   colorOffset--; nop;
49   nop;}

```

Wird nun die Funktion `start_Recursion()` aufgerufen und ausgeführt, so ergibt sich die in Abbildung 6 dargestellte Ausgabe (`recurse(3)`). Dort sind Ähnlichkeiten zur Iteration zu erkennen: zunächst werden die Zeilen 42 bis 46 wiederholt ausgeführt (Aufbau der rekursiven Aufrufe); danach werden die Zeilen 46 bis 49 wiederholt ausgeführt (Abbau/Beendigung der rekursiven Aufrufe). Es ist zu sehen, wie jeweils ein Aufruf von `recurse()` durch einen weiteren Aufruf von `recurse()` unterbrochen wird. Besonders beim Aufruf `recurse(1)` (Schritte 13 bis 27), welcher durch den Aufruf `recurse(0)` (Schritte 19 bis 23) unterbrochen wird. Außerdem ist deutlich, dass die Schritte 19 bis 23 (lila) ein anderes Muster bilden, als die Schritte der Aufrufe mit den Argumenten 3, 2 und 1 (grün, gelb, blau). Der wesentliche Unterschied zwischen Rekursion und Iteration ist: Der wiederholt ausgeführte Code hat bei Iteration Zugriff auf denselben einzigen Kontext (stack frame, Aktivierungskontext); bei Rekursion gibt es mehrere verschiedene Kontexte (der selben Art).

Recursion

```

41
42 void r1curse(int t7rns){color013set++; nop19
43   nop; 2 8 14 20
44   nop; if(3u4ns>0) { n9p10 1516 21
45     nop;5 11 17
46     nop; r6curse(turns12 1); nop; 18 24 28 32
47     nop;} 25 29 33
48   nop; 22 26 30 34
49   nop;colorOffset--;} 23 27 31 35
50

```

Abbildung 6: Rekursion. Erhöhen bzw. Erniedrigen von `colorOffset` sorgt für Farbänderung.

2.2 Micro Style Guide

In späteren Materialpaketen wird auf die Themen Style Guides und best Practices eingegangen. Vorab jedoch kurz die wichtigsten Regeln in einem Beispiel:

```

1 void printLines(int numberOfLines){ // names should speak for themselves
2   int i = 0; // always initialize
3   if (numberOfLines > 0) { // use 2 or 4 spaces for indentation
4     // ...
5   } else {
6     int k = numberOfLines; // use small scopes for variables
7     while (k > 0) { // short names: counters, indices, etc.
8       // ...
9     } // clearly: end of while block
10  }
11 }
12
13 // have some space beetwen definitions
14 int main(){

```

Obiger Quelltext verwendet für *CamelCase* Bezeichner, wie es in der Java-Welt üblich ist. Viele C/C++-Programmierer verwenden jedoch *snake_case* für Bezeichner. Üblicherweise wird der zu verwendende Stil von der Organisation, die das Produkt entwickelt, vorgegeben.

Viele Editoren unterscheiden bei Kommentaren zwischen `//` und `///` sowie `/*` und `/**`, so dass sich Dokumentationskommentare typographisch anders darstellen lassen als Auskommentierungen.

Ein schlecht formatierter Quelltext läßt sich mit dem Kommando `clang-format --style=Mozilla main.c` umformatieren.

2.2.1 (Level C) Goto, considered harmful

Es wird davon abgeraten, `goto` zu verwenden². Es wird hier dennoch erwähnt, um als abschreckendes Beispiel dienen. Dies verdeutlicht wie wichtig es ist, Quellcode so zu gestalten, dass er lesbar und leicht verständlich ist.

```

111 void start_GotoHarmful(void){nop;
112     nop; goto label1;
113 label2: nop; goto label5;
114 label1: nop;
115     nop; goto label7;
116     nop;
117 label5: nop; goto lexit;
118 label7: nop; goto label2;
119     nop;
120 lexit:  nop;
121     nop;
122 }
```

Wird nun die Funktion `start_GotoHarmful()` ausgeführt, so ergibt sich die in Abbildung 7 dargestellte Ausgabe. Die Verwendung von `goto` ist sehr kontrovers und hat sich oft als unsicher herausgestellt (siehe “Hintergründe zur schweren SSL-Sicherheitslücke bei Apple”³ und “Linux kernel coding style”⁴).

```

Terminal — -bash — 76x15
goto (considered harmful)
111
112 void start_GotoHarmful(void){nop;
113     2op; goto label1;
114 label2: nop; goto label5;
115 label1: nop;
116     nop;4goto label7;
117     nop;
118 label5: nop; goto 7lexit;
119 label7: nop; goto 5label2;
120     nop;
121 lexit:  nop;          8
122     nop;          9
123 }
```

Abbildung 7: Das von C übernommene `goto` verdeutlicht, wie nah C und C++ an Assembler heranreichen.

²Es gibt einige wenige Einsatzzwecke für `goto`. Beispielsweise macht der Code des Linux-Kernels häufig Gebrauch von `goto fail`; ebenso ist es möglich, Kontrollstrukturen umzusetzen, die C++ nicht unterstützt.

³Hintergründe zur schweren SSL-Sicherheitslücke bei Apple <https://m.heise.de/mac-and-i/artikel/Hintergruende-zur-schweren-SSL-Sicherheitsluecke-bei-Apple-2121951.html?seite=all>

⁴Linux kernel coding style: <https://www.kernel.org/doc/html/v4.17/process/coding-style.html>

Die oben vorgestellten Kontrollstrukturen lassen sich mit `if` und `goto` nachbilden. So wird das jeweilige Verhalten explizit sichtbar.

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

Wichtig: Nehmen Sie die Datei `mods.cpp` als Ausgangspunkt für ihr Grundgerüst: erstellen Sie hieraus eine Kopie `mod_0.cpp` in der sich neben `main()` und `performPattern()` lediglich eine Funktion `void mod_0(void)` befindet (analog zu `start_Sequence()`). Nachdem Sie sichergestellt haben, dass die Datei fehlerfrei übersetzt wird und funktioniert, erstellen Sie die Kopien `mod_1.cpp`, `mod_2.cpp` und `mod_3.cpp`, welche Sie dann zum Üben verwenden können. Zum Testat bringen Sie jedoch die leeren Dateien mit (Kopien von `mod_0.cpp`).

Beachten Sie, dass Sie bei der Ausführung gegebenenfalls ein sehr großes Terminal-Fenster benötigen und einen Buchstaben eingeben müssen, um das Programm zu beenden.

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. Lassen Sie in `mod_1.cpp` mit dem `nop`-Makro folgendes Muster erzeugen:

```

10
11     1
12     2
13     3
14     4 5 6 7 8 9 10 11 12 13 14 15
15                                     16
16                                     17
17

```

2. Lassen Sie in `mod_2.cpp` mit dem `nop`-Makro folgendes Muster erzeugen:

```

18
19     1
20     2
21     3
22
23     4 5 6 7 8 9
24
25     10 11 12 13 14 15
26
27                                     16
28                                     17
29

```

3. Lassen Sie in `mod_3.cpp` mit dem `nop`-Makro folgendes Muster erzeugen:

```

    4 5 6 7 8 9 10 11 12 13
1
2
3
14
15
16

```

4. Wählen Sie eine der Drei Aufgaben aus:

- Implementieren Sie eine Funktion `bool isLeapYear (unsigned int year)`, welche für die Jahre 1582 bis 2052 zurückliefert, ob es sich um ein Schaltjahr handelt. Hinweis: Mit dem Ausdruck `(y % 200) == 0` können Sie herausfinden, ob `y` durch 200 teilbar ist. Verwenden Sie `if`-Statements und `if-else`-Statements statt logischer Operatoren.
- Erstellen Sie eine Funktion `void FizzBuzz(int max)`, welche für jeden Wert von 1 bis `max` die jeweilige Zahl ausgibt. Ist die Zahl jedoch durch drei teilbar, wird `Fizz` ausgegeben. Ist die Zahl jedoch durch fünf teilbar, wird `Buzz` ausgegeben.
- Erweitern Sie die Funktion `FizzBuzz()` so, dass `FizzBuzz` ausgegeben wird, sofern die Zahl durch drei und durch ist.

- Ändern Sie die Logik des obigen Programmes so, dass im dritten Fall nicht **FizzBuzz**, sondern **Buzzer** ausgegeben wird.
 - Implementieren Sie das Engineering Flow Chart⁵ in Form einer Funktion, die ungefähr diese Signatur hat: `void printToolFor(string name, bool moves, bool shouldMove);`
5. (Level C) Lassen Sie mit dem `nop`-Makro folgendes Muster erzeugen (Hinweis: die Schritte 3 bis 12 sind in einer anderen Subroutine, welche eine Selektion beinhaltet):

```

32
33     1
34     2
35
36                13
37
38
39         3   5   7   9   11
40
41         4       8       12
42
43             6       10
44

```

9. (Level C) Tragen Sie die durchnummerierten Schritte in diesen Code ein:

```

bool isBalanced(std::string s){nop;
    int braceCount = 0; nop;
    nop;for(size_t i=0;i<s.length();i++){
        nop; if(s[i] == '('){
            braceCount++; nop;
        }else if(s[i] == ')'){
            braceCount--; nop;
        }
    }
    nop; return braceCount == 0;
}

void start_exercise4(void){nop;
    nop; if(isBalanced("a(bc)d")){
        nop; // fine!
    }else{
        nop;
    }
    nop;}

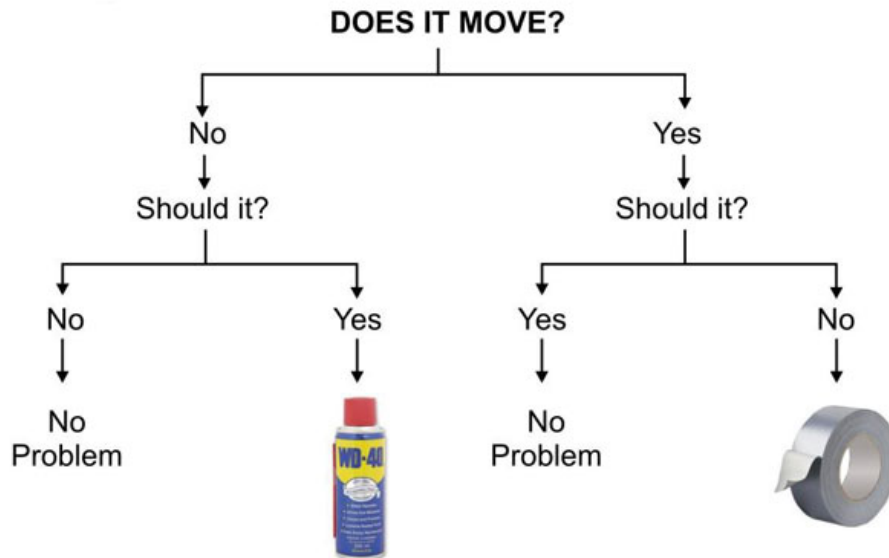
```

10. (Level C) Bauen Sie eine Funktion `recurse()`, welche sich selbst aufruft.

⁵<https://waynedalenews.com/2018/10/duck-or-duct-tape-homeowner-rx/>

Für jeden Aufruf soll der Rückgabewert von `polynom1()` ausgegeben werden. In `main()` wird `recurse(20)` aufgerufen; `recurse()` ruft sich also selbst mit den Werten 19 bis 0 auf

11. (Level C) Geben Sie während der obigen Rekursion die Werte in umgekehrter Reihenfolge aus, jedoch ohne die Aufrufparameter und die dazugehörige Logik zu verändern



Engineering Flowchart.

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. Stellen Sie sich eine Sequenz von gleichartigen Anweisungen vor (z.B. `arr[0] = 0; arr[1] = 0; arr[2] = 0; etc.`). Welche Vorteile hat es, hier stattdessen eine Iteration zu verwenden?
2. Welche Vorteile hat es, eine Anweisungssequenz in einer Funktion zu bündeln?
3. Welche Vorteile hat es, eine Problemstellung rekursiv zu implementieren?
4. Welche Nachteile hat es, eine Problemstellung rekursiv zu implementieren?
5. Was macht die Schwierigkeit bei der Programmierung des `FizzBuzz`-Spieles aus?
6. Warum sollten Style Guides eingehalten werden? Was sind die Vorteile des

oben skizzierten Micro Style Guides?

7. Freiwillig: Zeigen Sie auf, inwiefern das Piktogramm auf der Titelseite dieses Materialpaketes den Inhalt zusammengefasst darstellt.
8. **Level C:** Betrachten den nachfolgenden Code (siehe [SEdge]) und beantworten diese Fragen (Code-Ausführung in Gedanken simulieren):
 - Welche Ausgabe ergibt sich bei `exec state 2` für den Aufruf `gcd(16,24)`?
 - Welche Ausgabe ergibt sich bei `exec state 3` für den Aufruf `gcd(23, 13)`?

```
1 unsigned int gcd(unsigned int a, unsigned int b){
2     if( a < b ){
3         std::swap( a, b );
4     }
5     int loopnum = 0;
6     while( b > 0 ){
7         unsigned int r = a % b;
8         loopnum++; println("exec state ", loopnum, " : ", " a=", a, " b=", b, " r=", r);
9         a = b;
10        b = r;
11    }
12    return a;
13 }
```

4 Nützliche Links

- GeeksForGeeks, Insertion Sort: <https://www.geeksforgeeks.org/insertion-sort/>
- Hacker News, About coding the FizzBuzz interview question: <https://news.ycombinator.com/item?id=20786922>
- C2 Wiki, Fizz Buzz Test: <https://wiki.c2.com/?FizzBuzzTest>

5 Literatur

- [SEdge] Robert Sedgewick, Kevin Wayne: Algorithms
- [PPP] Stroustrup, Bjarne: Programming - Principles and Practice using C++
- [TCPL] Stroustrup, Bjarne: The C++ Programming Language, Fourth Edition