Materialpaket 03_FLOW_a – Control Flow

C/C++, Autor: Prof. Dr.-Ing. Carsten Link

Version 1.3.1 March 3, 2019

Contents

1 Kompetenzen und Lernegebnisse

Durch das Bearbeiten dieses Materialpaketes erwerben Sie diese Kompetenzen (Wissen, Fähigkeiten, 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. Lernegebnisse: Sie können nachweislich[1]:

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

- 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 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.

2.1 Grundlegender Kontrollfluss

Im folgenden werden zwei Makros verwendet, die helfen, den Kontrollfluss in einem Programm sichtbar zu machen. Der Makro `INITPRINT("Titel");` initialisiert globale Variablen und setzt eine Überschrift. Der Makro `PRINT` sorgt dafür, dass die aktuelle Zeilennummer ausgegeben wird. Jede Verwendung des Makros setzt die Schreibposition um zwei Zeichen nach Rechts (eine Art Zeitachse). Die y-Position der Schreibposition ergibt sich aus der aktuellen Zeilennummer (links oben wird mit der Zeilennummer der letzten `INITPRINT`-Verwendung begonnen).

Auf diese Weise werden nun die ausgeführten Zeilennummern über der Zeit dargestellt.

2.1.1 Sequenz

```
30 void start_Sequence(void){
31   INITPRINT("Sequence");
32   PRINT;
33   PRINT;
34   PRINT;
35   PRINT;
36   PRINT;
37   PRINT;
38   PRINT;
39   PRINT;
40   PRINT;
41   PRINT;
42 }
```

Wird nun die Funktion `start_Sequence()` aufgerufen und ausgeführt, so ergibt sich die folgende Ausgabe:

```
Sequence
32
 33
   34
     35
       36
         37
           38
             39
               40
                 41
```

Zu sehen sind die Ausgeführten Zeilennummern (32 bis 41), wobei mit jeder Verwendung von `PRINT` die Ausgabe nach Rechts verschoben wird, so dass der logische/zeitliche Ablauf erkennbar ist.

2.1.2 Selektion

```
50 void start_SeqSelection(void){
51   INITPRINT("Selection");
52   PRINT;  int b = 1;
53   PRINT;  if (b == 1){
54     PRINT;
55     PRINT;
56     PRINT;
57     PRINT;
58     PRINT;  }else{
59     PRINT;
60     PRINT;
61     PRINT;
62     PRINT;
63     PRINT;
64             }PRINT;
65   PRINT;
66 }
```

Wird nun die Funktion `start_SeqSelection()` aufgerufen und ausgeführt, so ergibt sich die folgende Ausgabe:

```
Selection
52
 53
   54
     55
       56
         57
           58




             64
               65
```

Oben wird ersichtlich, dass die Zeilen im `if`-Zweig ausgeführt wurden (da die dortige Bedingung erfüllt war), die des `else`-Zweiges jedoch übersprungen wurden (die umgekehrte Bedingung war ja nicht erfüllt).

2.1.3 Iteration

```
70  void start_Iteration(void){
71    INITPRINT("Iteration");
72    PRINT; int i=9;
73    PRINT; while(i>=0){
74      PRINT; i--;
75    }PRINT;
76    PRINT;
77  }
```

Wird nun die Funktion `start_Iteration()` aufgerufen und ausgeführt, so ergibt sich die folgende Ausgabe:

```
Iteration
72
 73
   74747474747474747474
                        75
                          76
```

In der Ausgabe ist zu erkennen, dass die Zeile `74` mehrfach ausgeführt wird, da sie sich in dem Block der `while`-Iteration befindet. Die Bediungung für die Wiederholung ist `10` mal erfüllt.

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.

```
80  int func_1(int arg);
81
82  // Sequence, Selection, Iteration, Subroutine
83  void start_subroutine(void){
84    INITPRINT("Subroutine");
85    PRINT; func_1(1);
86    PRINT;
87  }
88
89
90  int func_1(int arg){
91    PRINT; int local_1;
92    PRINT;
93    PRINT;
94    PRINT;
95    PRINT; return arg * local_1;
96  }
```

Wird nun die Funktion `start_subroutine()` aufgerufen und ausgeführt, so ergibt sich die folgende Ausgabe:

```
Subroutine
85
            86




  91
    92
      93
        94
          95
```

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.

2.1.5 Rekursion

```
9  void recurse(int); // forward declaration because of firstLine initialization
10 void start_Recursion(){
11   INITPRINT("Recursion");
12   PRINT;recurse(10);
13   PRINT;
14 }
15 void recurse(int turns){
16   PRINT; if(turns>0){
17     PRINT;  recurse(turns - 1);
18   }
19   PRINT;
20 }
```

Wird nun die Funktion `start_Recursion()` aufgerufen und ausgeführt, so ergibt sich die folgende Ausgabe:

```
Recursion
12
                                                                    13


 16  16  16  16  16  16  16  16  16  16  16
   17  17  17  17  17  17  17  17  17  17

                                                  191919191919191919191919
```

Oben sind Ähnlichkeiten zur Iteration zu erkennnen: zunächst werden die Zeilen `16` und `17` wiederholt ausgeführt (Aufbau der rekursiven Aufrufe); danach wird die Zeile `19` wiederholt ausgeführt (Abbau/Beednigung der rekursiven Aufrufe). Der wesentlicher 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).

3 Material zum aktiven Lernen

Da eine Programmiersprache nur durch aktive Verwendung erlernt werden kann, werden im folgenden Aufgaben zum praktischen Üben vorgestellt. Zunächst wird ein Grundgerüst (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

Nehmen Sie die Datei `main_mp2_FLOW_a.cpp` als Grundgerüst. 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

Stellen Sie sicher, dass Sie jede einzelne der nachfolgenden Modifikationen innerhalb weniger Minuten (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.

Modifikationen:

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

```
12
                          13


 161616161616

             18181818181818
```

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

```
15          15
 16           16
   17171717     17171717
```

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 Problemstellung rekursiv zu implementieren?
3. Welche Nachteile hat es, eine Problemstellung rekursiv zu implementieren?

4 Literatur

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