C/C++ Materialpaket (Level C)
03a_DATA – Variables and Values

Prof. Dr. Carsten Link

Zusammenfassung

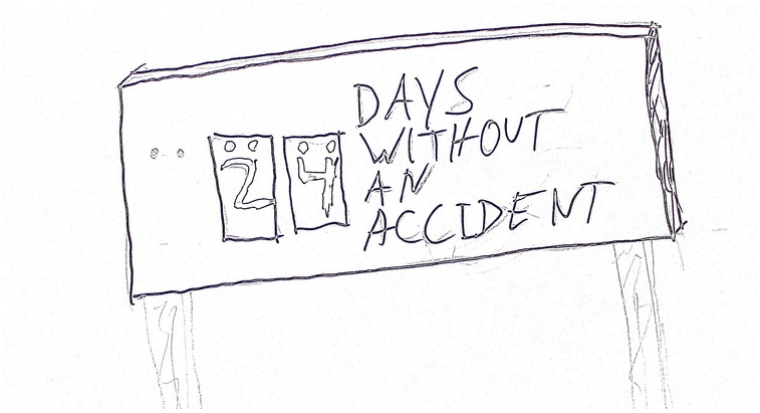

Abbildung 1: Platzhalter für Werte und was man damit machen kann

Inhaltsverzeichnis

1 Kompetenzen und Lernegebnisse **2**

2 Konzepte **2**
2.1 Begriffe 2
2.2 Boolsche Werte 3
2.3 Zahlenwerte 4
2.4 Texte 6
2.5 Implizite Typumwandlungen 7
2.6 Implizite Typumwandlungen zu bool 7
2.7 Explizite Typumwandlungen 8
2.8 Variablen 9
2.9 Sichtbarkeit von Variablen 9
2.10 Paramter und Rückgabewerte 10
2.11 Referenzen 10
2.11.1 Referenzen als Verweise auf einen einzigen Wert 10
2.11.2 (Level C) Referenzen als Rückgabewerte 11

3 Material zum aktiven Lernen **12**
3.1 Aufgabe: Grundgerüst 12
3.2 Aufgabe: Modifikationen 12
3.3 Verständnisfragen 13

4 Nützliche Links **14**

5 Literatur **15**

1 Kompetenzen und Lernegebnisse

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

Sie können die wichtigsten primitiven Datentypen benutzen und Werte eines Typs in einen anderen Typ umwandeln lassen.

Die oben genannten Kompetenzen erwerben Sie, indem Sie Lernziele erreichen, welche sich prüfen lassen. Lernegebnisse: Sie können nachweislich[1]:

- die Begriffe Literal, Wert einordnen
- Bool'sche Werte in C-Programm berechnen lassen
- explizite und implizite Umwandlungen zwischen Datentypen vornehmen oder vornehmen lassen

2 Konzepte

In den folgenden Unterabschnitten werden die Konzepte vorgestellt, welche Sie im Abschnitt *Material zum aktiven Lernen* praktisch umsetzen können.

2.1 Begriffe

- **Wert:** ein Bitmuster im Speicher, das als Ausprägung eines Typens interpretiert wird
- **Literal:** eine Zeichenkette im Quelltext, die einen konkreten Wert darstellt Beispielsweise stellt `0.0` die Fließkommazahl mit dem Zahlenwert Null dar und das Literal `false` stellt den Wahrheitswert falsch dar
- **Variable:** auf Speicherstellen für Werte kann jeweils über einen Namen zugegriffen werden. Im Verlauf der Materialpakete werden diese Begriffe erläutert: lokal, global, Sichtbarkeit, Gültigkeit, Initialisierung, Speicherort, Namensgebung

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

2.2 Boolsche Werte

Der Programmablauf ist in der Regel von den verarbeiteten Daten abhängig. In den nachfolgenden Unterkapiteln werden Kontrollstrukturen vorgestellt, mit der sich der Ablauf eines Progammes datenabhängig steuern lässt (beispielsweise `if` und `while`). Diese Kontrollstrukturen verwenden boolsche Ausdrücke, die in einem Wahrheitswert münden, der wiederum direkt den Ablauf beeinflusst. Der nachfolgende Code illustriert dies:

```
4   int seven         = 7;
5   int eleven        = 11;
6   char zulu         = '\0';
7
8   bool b_true1      = (true);
9   bool b_true2      = (!false);
10  bool b_smaller    = (seven < eleven);
11  bool b_int        = (seven);
12  bool b_eqal       = (seven == 7);
13
14  bool b_false1     = false;
15  bool b_false2     = !true;
16  bool b_char       = zulu;
17  bool b_greater    = seven > eleven;
18  bool b_neqal      = seven != 7;
```

Oben wird die Variable `b_true1` mit dem `bool`-Literal `true` initialisiert. Der Variablen `b_true2` hingegen wird der Wert des Ausdrucks `!false` entpsricht (Negierung des Wertes des Literals `false`).

In den Zeilen `10`, `12`, `17`und `18` zeigt sich, dass Vergleichsoperatoren (dort `<`, `==`, `>`und `!=`) zu einem `bool`-Wert ausgewertet werden, der dann jeweils zugewiesen wird.

Es ist weiterhin zu sehen, dass in C/C++ auch `char` und `int`-Werte als Wahrheitswerte interpretiert werden können. Hiebei entspricht der Wert `0` dem Wert `false`; alle anderen Zahlenwerte ergeben `true`. Ebenso wird deutlich, dass die logischen Ausdrücke eingeklammert werden können, um die Lesbarkeit zu erhöhen oder die richtige Auswertungsreihenfolge sicherzustellen.

Im nachfolgenden Code werden obige Wahrheitswerte verwendet:

```
20  bool b_complex    = (seven < eleven) && (5>3) && (zulu != 'z');
21
22  bool allTrue      = b_true1   &&  b_true2  && b_smaller
23                      &&  b_int && b_eqal && b_complex;
24
25  bool allFalse     = !b_false1  && !b_false2 &&  !b_char
26                      && !b_greater && !b_neqal;
27
```

```
28   //bool allFalse2  = !(b_false1 ||   b_false2 ||   b_char
29   //                     ||  b_greater ||  b_neqal);
30
31   if (allTrue && allFalse /* && allFalse2*/) {
32     println("all fine!");
33     return 0;
34   } else {
35     println("error!");
36     return 1;
37   }
```

Bei der Zuweisung zur Variablen `b_complex` ist zu sehen, dass Bedingungen beliebig kombiniert werden können. Es wird empfohlen, kombinierte Bedingungen *immer* zu klammern.

2.3 Zahlenwerte

In diesem Abschnitt werden die wichtigsten primitiven Datentypen behandelt. Die Wertebereiche der wichtigsten primitiven Datentypen sind wie folgt:

- `char`: Zeichen, d. h. Buchstaben (`A ... Z, a ... z`), Ziffern `0 ...9`, Sonderzeichen (`$%&/()`) sowie Steuerzeichen wie beispielsweise Tabulator oder Zeilenumbruch
- `unsigned char`: Zahlen im Bereich `0 ... 255` (meist 8 Bit, `uint8_t`[2])
- `int`: für Ganzzahlen mit Vorzeichen im Bereich -2×10^9 … 0 … 2×10^9 (`int32_t`)
- `unsigned int`: für positive Ganzzahlen 0 … 4×10^9 (`uint32_t`)
- `long`: für Ganzzahlen mit Vorzeichen -9×10^{18} … 0 … 9×10^{18} (`int64_t`)
- `unsigned long`: für positive Ganzzahlen 0 … 1.8×10^{19} (`uint64_t`)
- `double`: für Fließkommazahlen mit 15 Dezimalstellen Genauigkeit[3]

Die konkreten Wertebereiche von Ganzzahl- und Fließkommazahl-Datentypen sind CPU-, Compiler-, und Betriebssystem-abhängig[4]. Das folgende Programm verdeutlicht, wie die Grenzen der Wertebereiche ermittelt werden können (der `sizeof`-Operator liefert die Größe seines Operanden in Bytes zurück):

```
1  #include "println.hpp"
2  #include <limits>
3  #include <cstdint>
4
5  #define lowest_of(x) std::numeric_limits<x>::lowest()
6  #define max_of(x) std::numeric_limits<x>::max()
7
```

[2] aus `<cstdint>`

[3] d. h. 14 Nachkommastellen in wissenschaftlicher Darstellung. Fließkommazahlen bereiten vielfältige Probleme, die in diesem Kurs nicht betrachtet werden.

[4] siehe `limits`-Header und http://en.cppreference.com/w/cpp/types/numeric_limits

```
8  int main()
9  {
10   println("=========================================================");
11   println("type\t|sizeof\t|\t lowest\t\t|\t highest");
12   println("---------------------------------------------------------");
13   println("int   \t| ",sizeof(int), "\t|\t",
14           lowest_of(int), "\t|\t",
15           max_of(int));
16   println("long  \t| ",sizeof(long), " \t|\t",
17           lowest_of(long), "\t|\t",
18           max_of(long));
19   println("uint32_t| ",sizeof(uint32_t), "\t|\t\t",
20           lowest_of(uint32_t), "\t|\t",
21           max_of(uint32_t));
22   println("float \t| ",sizeof(float), "\t|\t",
23           lowest_of(float), "\t|\t",
24           max_of(float));
25   println("double\t| ",sizeof(double), "\t|\t",
26           lowest_of(double), "\t|\t",
27           max_of(double));
28   println("=========================================================");
29 }
```

Die Ausgabe könnte wie folgt aussehen (MacOS Intel 64 Bit):

```
=========================================================
type    |sizeof |    lowest         |     highest
---------------------------------------------------------
int     | 4     |   -2.147484e+09   |   +2.147484e+09
long    | 8     |   -9.223372e+18   |   +9.223372e+18
uint32_t| 4     |        0          |   4.294967e+09
float   | 4     |   -3.402823e+38   |   3.402823e+38
double  | 8     |   -1.797693e+308  |   1.797693e+308
=========================================================
```

Die Ausgabe könnte wie folgt aussehen (Rapsberry Pi 32 Bit):

```
=========================================================
type    |sizeof |    lowest         |     highest
---------------------------------------------------------
int     | 4     |   -2.147484e+09   |   +2.147484e+09
long    | 4     |   -2.147484e+09   |   +2.147484e+09
uint32_t| 4     |        0          |   4.294967e+09
float   | 4     |   -3.402823e+38   |   3.402823e+38
double  | 8     |   -1.797693e+308  |   1.797693e+308
=========================================================
```

Da diese Werte von der Hardwarearchitektur, dem Betriebsystem und dem Compiler abhängen können, ist es gelegentlich sinnvoll, integrale Datentypen mit fest vorgegebener Größe zu verwenden. So finden sich in `<stdint.h>` neben `int16_t` für 16-Bit breite vorzeichenbehaftete Ganzzahlen und `uint64_t` für 64-Bit breite positive Ganzzahlen noch viele weitere Typdeklarationen und Makro-Konstanten wie z. B. `INT32_MIN`.

Hinweis: das obige Programm verwendet aus der Header-Datei `helpers/println.hpp` die spezielle Funktion `println()`, welche einfacher für textuelle Ausgaben zu verwenden ist, als die üblichen C++-Standardausgabeverfahren (Streams mit `std::cout`). Beispiele hierzu finden sich in der Datei `z_printlnDemo.cpp`. Darin ist zu sehen, wie die Funktion mit mehreren verschiedenen Parametertypen aufgerufen wird.

2.4 Texte

Eine genauere Betrachtung von Zeichenketten findet sich im nächsten Materialpaket. Hier soll nur eine kleine Übersicht gegeben werden.

C++ hat von C Zeichenketten übernommen. Da der Umgang mit C-Strings sehr umständlich und fehleranfällig ist, findet sich in der C++-Standardibliothek der Typ `std::string`. Dieser kann mit C-String-Literalen (beispielsweise `"Hello, world!"`) initialisiert werden und verfügt über komfortable Operationen:

```
#include "println.hpp"

int main()
{
  std::string hello = "Hello";
  std::string world = "world";
  std::string greeting = hello + ", " + world + '!';
  println("greeting=", greeting);
  if(hello == "Hallo"){
    println("Hallo == ", hello);
  }else{
    println("Hallo != ", hello);
  }
  std::string a = "Aachen";
  std::string z = "Zwickau";
  std::string cmp;
  if (a < z){
    cmp = " before ";
  }else{
    cmp = " after ";
  }
  println(a, cmp, z);
}
```

2.5 Implizite Typumwandlungen

Der Compiler kann eine implizite Typumwandlung vornehmen, falls bei einer Zuweisung oder für einen Funktionsparameter ein Wert eines anderen Typens benötigt wird, als der Typ des angegebenen Ausdrucks. Der folgende Code zeigt einige dieser impliziten Typumwandlungen:

```
void implicitConversions(){
  // non-narrowing; integral promotion
  unsigned char c = 127;
  unsigned int  i = c;
  unsigned long l = i;
  println("implicit: c=", c, " l=", l);
  // narrowing
  unsigned long ln = 1024 + 127;
  unsigned int  in = ln;
  unsigned char cn = in;
  println("implicit: ln=", ln, " cn=", cn);
}
```

Der obige Code erzeugt folgende Ausgabe:

```
implicit: c=127 l=127
implicit: ln=1151 cn=127
```

An der Ausgabe ist zu sehen, dass `1151` in der letzten Zeile zu `127` verfälscht wurde, da `unsigned char` einen kleineren Wertebereich hat, als `unsigned int`. Jedoch sind bei dieser `unsigned`-Umwandlung die niederwertigsten Bits erhalten geblieben.

Es wird empfohlen, implicit narrowing conversions zu vermeiden und stattdessen mit einer expliziten Typumwandlung die Intention klarzustellen. Siehe dazu das nächste Unterkapitel.

2.6 Implizite Typumwandlungen zu bool

Werte der integralen Typen (`int`, `char`, etc.) können vom compiler implizit in `bool`-Werte umgewandelt werden. Hierbei entsprechen die Nullwerte `false` und alle anderen Werte werden in den `bool`-Wert `true` umgewandelt.

```
void implicit_bool(){
  bool yes = true;
  if (yes) {
    println("yes is true");
  }
  char c = '\n';
  if (c) {
    println("c is truthy");
  }
```

```
  int  i = 17;
  if (i) {
    println("i is truthy");
  }
  long zero = 0;
  if (zero) {
    println("error!");
  }else{
    println("is not truthy");
  }
}
```

Der obige Code zeigt, dass `c`, `i`, und `zero` implizit zu `bool`-Werten umgewandelt werden.

Achtung! Obwohl `if (yes) { }` und `if (yes == true) { }` wie erwartet funktionieren, ist das bei `if (yes == i) { }` nicht der Fall. Hierbei wird vom Compiler `yes` implizit in den Integer `1` umgewandelt. Der Vergleich `1==17` liefert den Wert `false`.

2.7 Explizite Typumwandlungen

Um vom Compiler eine Typumwandlung explizit anzufordern, wird mit `static_cast<T>(expr)` der Wert des Ausdrucks `expr` in einen Wert vom Typ `T` umgewandelt. Hier einige Beispiele:

```
// using unsigned here, since narrowing signed is implementation-defined
void exlicitConversions(){
  //non-narrowing
  unsigned char c = static_cast<unsigned char>(127);
  unsigned int  i = static_cast<unsigned int>(c);
  unsigned long l = static_cast<unsigned long>(i);
  println("exlicit: c=", c, " l=", l);
  //narrowing
  unsigned long ln = static_cast<unsigned long>(1024 + 127);
  unsigned int  in = static_cast<unsigned int>(ln & 0xFFFFFFFF);
  unsigned char cn = static_cast<unsigned char>(in & 0xFF);
  println("exlicit: ln=", ln, " cn=", cn);
}
```

In obigem Code sind unterhalb des Kommentars `//non-narrowing` Umwandlungen angegeben, bei denen der Quellwert in die Zielvariable passt, so dass keine Probleme auftreten können.

Bei den Umwandlungen unterhalb des Kommentars `//narrowing` hingegen sind Umwandlungen zu sehen bei denen Probleme auftreten können. Hier bieten sich verschiedene Moglichkeiten der Handhabung an:

- die Programmlogik stellt sicher, dass die Werte sich im Wertebereich des Zieltyps befinden
- es werden bewusst nur einzelen Bits übernommen (z. B. `c = ln & 0xFF`)
- Fehlerbehandlung

Für die Umwandlung von Zahlen, die als Zeichenketten gegeben sind, in Zahlenwerte (und umgekehrt) stehen diese Funktionen zur Verfügung:

- `std::stoi()`: string to `int`
- `std::stol()`: string to `long`
- `std::stoul()`: string to `unsigned long`
- `std::stod()`: string to `double`
- `std::to_string()`: wandelt ganzahlige Werte und Fließkommawerte in eine `std::string`-Zeichenkette

Hinweis: Die Verwendung der hier nicht vorgestellten C-artigen Typumwandlungen (`(T)expr` und functional-style `T(expr)`) sollte vermieden werden, da diese umgewollte Effekte haben können.

2.8 Variablen

Der Begriff Variable umfasst einen Namen für eine Speicherstelle, die einen Wert aufnehmen kann. Der Wert in der dem Namen zugeordneten Speicherstelle kann geändert werden. Die erste Änderung wird Initialisierung genannt.

Die beiden wichtigen Eigenschaften Sichtbarkeit und Lebenszyklus werden im nächsten Unterkapitel bzw. im Materialpaket 05c_OO_CYCL vorgestellt.

2.9 Sichtbarkeit von Variablen

```
1  #include "println.hpp"
2
3  int global;
4
5  void foo(int param){ // param is local var, but initialized automatically
6    int local=7;
7    global = param;
8    global = local;
9    for(int i=0;i<1;++i){
10     global = param;
11     global = local;
12     global = i;
13     {
14       int k=13;
15       global = k;
16     }
17     //global = k; // k is not visible here
```

```
  }
  //global = i; // i is not visible here
}

int main()
{
  global = 11; // local, param, i, k are not visible here
  foo(9);
  println("s");
}
```

2.10 Paramter und Rückgabewerte

```
#include "println.hpp"

void call_by_value(int seven){
  // "int seven" is a local variable, which is
  // initialized with a COPY of caller's value (e.g. copy of 7)
  seven = seven + 1;
}

int return_by_value(void){
  int result = 11;
  return result; // returns a COPY of result's value (e.g. copy of 11)
}

int main()
{
  int seven = 7;
  call_by_value(seven);
  println("seven=", seven);
  int eleven = return_by_value();
  println("eleven=", eleven);
}
```

2.11 Referenzen

Referenzen sind Verweise auf Werte.

2.11.1 Referenzen als Verweise auf einen einzigen Wert

Mit Referenzen ist es möglich, mehrere Namen für eine einzige Speicherstelle (einen einzigen Wert) zu verwenden:

```
1  #include "println.hpp"
2
3  int main()
4  {
5    int   value = 0;
6    int & reference = value;
7    int & alias     = value;
8    int & spitzname = value;
9    int & verweis   = value;
10
11   println("value before ", value);
12   value     = value + 1;
13   reference = reference + 20;
14   alias     =     alias + 300;
15   spitzname = spitzname + 4000;
16   verweis   =   verweis + 50000;
17   println("value after ", value);
18 }
```

Referenzen werden bei der Initialisierung mit dem Zielwert verbunden (Zeilen 6 bis 9). Alle weiteren Zuweisungen mit dem =-Operator beziehen sich auf den Wert (Zeilen 13 bis 16). Der obige Code erzeugt folgende Ausgabe:

```
value before +0
value after +54321
```

2.11.2 (Level C) Referenzen als Rückgabewerte

```
1  #include "println.hpp"
2  #include <cctype>
3
4  int alphaCount;
5  int numCount;
6  int other;
7
8  // note: returns a reference to an integer
9  // may be used on the left hand side of an assignment
10 int& counterForChar(char c){
11   if(isalpha(c)){
12     return alphaCount;
13   } else if(isdigit(c)){
14     return numCount;
15   } else {
16     return other;
17   }
18 }
```

```
19
20 void countChars(std::string& s){
21   for (size_t i=0; i<s.length(); ++i){
22     int& counter = counterForChar(s[i]);
23     counter += 1;        // or counterForChar(s[i]) += 1;
24   }
25   s="done."; // note: will modify caller's value
26 }
27
28 int main()
29 {
30   std::string hex("0123456789abcdef");
31   println("hex=", hex);
32   countChars(hex);
33   println("alphaCount=", alphaCount, " numCount=", numCount, " other=", other);
34   println("hex=", hex);
35 }
```

Der obige Code erzeugt folgende Ausgabe:

```
hex=0123456789abcdef
alphaCount=+6 numCount=+10 other=+0
hex=done.
```

Referenzen als Rückgabewerte spielen beim Überladen von Operatoren eine wichtige Rolle (siehe `04_UDEF`). Dort kann mit Überladung des Subcript Operators (`[]`) eine Array-artige Nutzung ermöglicht werden.

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

Nehmen Sie die Dateien `z_printlnDemo.cpp` und `z_build.sh` als Vorlage und erstellen eigene angepasste Dateien, in denen Sie mit der Ausgabefunktion `println()` experimentieren.

3.2 Aufgabe: Modifikationen

1. Erweitern Sie das Grundgerüst um eine Funktion `void printInfoFor( std::string town, unsigned int population, bool hasUniversity)`,

welche Ausgaben der Art "Die Stadt Leer hat 35000 Einwohner …" macht. Die Ausgabe soll mit der Zeile `println(infoString)` erfolgen (daher müssen Sie `population` und `hasUniversity` in einen `std::string` umwandeln).

2. (Level C) Experimentieren Sie mit verschiedenen Möglichkeiten der Umwandlung des Zahlenwertes in der Funktion `printInfoFor()` (Stichworte `std::strstream` und `std::format`). Versuchen Sie auch, die Zahlen immer in einer festen Länge auszugeben (führende Nullen bzw. padding).
3. Addieren Sie die Rückgabewerte von `polynom1()` aus dem Materialpaket `01a_SIMPL` in einer `for` oder `while`-Schleife in `main()` auf
4. Ermitteln Sie, wie viele Nullstellen in dem abgesuchten Bereich von `polynom1()` aus dem Materialpaket `01a_SIMPL` liegen (d.h. Rückgabewert `== 0`), und lassen Sie die ermittelte Anzahl am Ende ausgeben. Erstellen Sie eine Funktion `numZeros()` dafür.
5. Implementieren Sie ein 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 ausschließlich logische Operatoren und Vergleichsoperatoren (`&&`, `||`, `!`, `==`, `!=`) sowie Klammern. Fangen Sie mit den einfachen Regeln an und erweitern so weit Sie können.
6. (Level C) Erweitern Sie das Grundgerüst um eine Funktion `isOfCharacterKind(char c, bool isAlphabetic, bool isNumeric, bool isUpperCase);` welche `true` liefert, falls alle Bedingungen entsprechned der aktuelle Parameter erfüllt sind. Verwenden Sie `isalpha()`, `isdigit()` , `isupper()` aus dem Bibliotheksheader `<cctype>`. Der Aufruf `isOfCharacterKind('X', true, false, true)` liefert demnach `true`. Beachten Sie, dass die Funktionen aus `<cctype>` keinen `bool`-Wert zurückliefern, was in Kombination mit impliziter Typumwandlung zu unerwünschtem Verhalten führt.
7. (Level C) Bauen Sie die Funktion `characterSequence(bool alphaLowerCase, bool alphaUpperCase, bool numeric)`, die einen `std::string` zurückliefert, welche alle durch die Flags aktivierten Zeichensequenzen enthält. Der Aufruf der neuen Funktion `charactersOfKind(true, false, true)` ergibt demnach `abcdefghijklmnopqrstuvwxyz0123456789`.

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. Was sind die Unterschiede und Gemeinsamkeiten zwischen den beiden Variablen `int i`, welche in verschiedenen Funktionen eine Programmes deklariert sind (beispielsweise eine in `main()` und eine in `polynom1()`)?
2. Welche Ausgabe erzeugt dieser Code?

```
int k;

void foo (){
  println(k);
}

int main(){
  int k = 7;
  foo();
  return 0;
}
```

3. Welches `i` wird geändert?

```
void foo (int i){
  i = i + 1;
}

int main(){
  int i = 7;
  foo(i);
  return 0;
}
```

4. Erläutern Sie auf welchen Arten `bool`-Werte entstehen können.
5. Was ist die genaue Ursache für Probleme mit dem Vergleich `isalpha('a') == true`?
6. Freiwillig: Zeigen Sie auf, inwiefern das Piktogram auf der Titelseite dieses Materialpaketes den Inhalt zusammengefasst darstellt.
7. (Level C) Wie können Sie obigen Code ändern, so dass das andere `i` geändert wird?
8. (Level C) Nehmen Sie an, Sie verwendeten keine Variablen; stattdessen nehmen Sie globale Variablen und fügen Präfixe an die Namen zur Unterscheidung (also statt `void foo(int i) {int k; ...` heißt es `int foo_para_i; int foo_local_k; void foo(){ ...`). Diskutieren Sie Gemeinsamkeiten und Unterschiede.

4 Nützliche Links

- TBD

5 Literatur

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