Programmierung 2 (SS 2021)
Universität des Saarlandes
Fakultät MI
Compiler Design Lab

Musterlösung Klausurvorbereitung

Prog2 Tutorenteam

Das Klausurvorbereitungsblatt wurde vom Prog2 Tutorenteam erstellt. Die Aufgaben wurden so selektiert, dass die Tutoren die Aufgaben als potentielle Klausuraufgaben einstufen oder sie auf solche vorbereiten. **Das ist keine Garantie, dass diese Aufgabentypen in der Klausur vorkommen!** Der Umfang des Blatts entspricht jedoch deutlich mehr als einer Klausur. Da die meisten Aufgaben komplett neu sind, können sich Fehler eingeschlichen haben. Fragt, falls ihr euch bei einer Lösung unsicher seid einfach im Forum nach. Wir versuchen dann die Aufgaben schnellstmöglich anzupassen.

Auf diesem Blatt wird es keine Schwierigkeitsgradangaben geben, da ihr diese in der Klausur auch nicht haben werdet und wir euch möglichst unbefangen an die Aufgaben herangehen lassen wollten. AuSSerdem setzen wir die Grundrechenarten, welche man im Arithmetikteil lernt für alle Aufgabenteile voraus, d.h. ihr solltet z.B. Hexadezimal- und Binärrechnung beherrschen.

Aufgabe 0: Kopieren > 7

Schreiben Sie eine MIPS-Prozedur, die alle Elemente > 7 aus einem Eingabepuffer in einen Ausgabepuffer kopiert und die Anzahl der kopierten Elemente zurückgibt.
Jedes Element ist 2 Bytes groß und wird vorzeichenlos interpretiert.
Sie dürfen annehmen, dass beide Puffer groß genug sind.
Ergänzen Sie hierfür folgenden Code:

```
1 # Argumente:
2 #   $a0: Addresse des Eingabepuffers.
3 #   $a1: Anzahl der Elemente im Eingabepuffer.
4 #   $a2: Addresse des Ausgabepuffers.
5 # Rückgabe:
6 #   $v0: Anzahl der kopierten Elemente
7 copy_greater_seven:
```

Lösung

Intuitiv wollen wir folgenden C-Code in MIPS schreiben (Annahme: 1 Byte = 8 Bit):

```
1 #include <stdint.h> /* Header für uint16_t (Vorzeichenlose, 16 Bit Ganzzahl) */
2
3 int copy_greater_seven(uint16_t* eingabepuffer /* $a0 */,
4                        int anzahl /* $a1 */,
5                        uint16_t* ausgabepuffer /* $a2 */)
6 {
7     register int kopiert = 0; /* $v0 */
8
9     while(anzahl){
10        register uint16_t cur = *eingabepuffer; /* $t0 */
11        if(cur > 7){
12           *ausgabepuffer = cur;
13           // da ein Eintrag 2 Byte groß ist, geht ++ 2 Byte weiter
14           ausgabepuffer ++;
15           kopiert ++;
16        }
17        // da ein Eintrag 2 Byte groß ist, geht ++ 2 Byte weiter
18        eingabepuffer ++;
19        anzahl --;
20    }
21    return kopiert;
22 }
```

MIPS-Code (zugehöriger C-Code ist rechts neben den Anweisungen ergänzt):

```
1  # Argumente:
2  #   $a0: Addresse des Eingabepuffers.
3  #   $a1: Anzahl der Elemente im Eingabepuffer.
4  #   $a2: Addresse des Ausgabepuffers.
5  # Rückgabe:
6  #   $v0: Anzahl der kopierten Elemente
7  copy_greater_seven:
8
9      # $v0 Setze Anzahl der kopierten Elemente auf 0
10     and $v0 $v0 $zero # register int kopiert = 0;
11
12     copy_greater_seven_loop:
13
14         # falls $a1 == 0 (alle Werte angeschaut), springe zum Ende
15         beqz $a1 copy_greater_seven_done # while(anzahl) {
16
17         # ------------------------------------------
18         #     Im Eingabepuffer sind noch Elemente,
19         #         die betrachtet werden müssen
20         # ------------------------------------------
21
22         # Lade aktuelles Element nach t0
23         lhu $t0 ($a0) # register uint16_t cur = *eingabepuffer;
24         # Überspringe kopieren, falls Element <= 7
25         ble $t0 7 copy_greater_seven_loop_copy_done # if(cur > 7) {
26
27             # ------------------------------------------
28             # Aktuelles Element ist > 7 (muss kopiert werden)
29             # ------------------------------------------
30
31             # Schreibe Element in Ausgabepuffer
32             sh $t0 ($a2)     # *ausgabepuffer = cur;
33             # Gehe in Ausgabepuffer eine Element (=2 Bytes) weiter
34             addiu $a2 $a2 2 # ausgabepuffer ++;
35             # Erhöhe Anzahl der kopierten Element um 1
36             addiu $v0 $v0 1 # kopiert ++;
37
38         copy_greater_seven_loop_copy_done: # }
39
40         # ------------------------------------------
41         #       Element wurde kopiert, falls nötig
42         # ------------------------------------------
43
44         # Schaue auf nächste Addresse im Eingabepuffer
45         addiu $a0 $a0 2 # eingabepuffer ++;
46         # Erniedrige die Anzahl der verbleibenden Elemente um 1
47         subiu $a1 $a1 1 # anzahl --;
48         # Springe zurück zur Bedingung der Schleife
49         b copy_greater_seven_loop # }
50
51     copy_greater_seven_done:
52
53     # ------------------------------------------
54     #     Alle nötigen Elemente kopiert, fertig!
55     # ------------------------------------------
56
57     # Springe zurück zum Aufrufer
58     jr $ra # return kopiert;
```

Aufgabe 1: Sportveranstaltung

Schreiben Sie ein C Programm, mit dem man Ergebnisse von Sportveranstaltungen auswerten kann. Sie bekommen dafür eine Datei gegeben, die folgendes Dateiformat hat:

- In der ersten Zeile ist die Anzahl an Teilnehmern gespeichert.
- Danach folgt für jeden Teilnehmer eine Zeile mit folgendem Format (mit Leerzeichen getrennt): Vorname (max. 63 Zeichen), Nachname (max. 63 Zeichen), Geschlecht („M“ / „W“ / „D“), Platzierung (Zahl > 0).

Für Ihr Programm können Sie davon ausgehen, dass das Dateiformat korrekt ist. Ihre Aufgabe ist nun folgendes:

1. Definieren Sie ein geeignetes `struct Teilnehmer`, in dem ein einzelner Teilnehmer mit seinen Daten gespeichert werden kann.
2. Implementieren Sie die Funktion `Teilnehmer* readTeilnehmer(FILE* fp)`, die einen Dateizeiger bekommt, und einen einzelnen Teilnehmer einliest. Die Funktion soll dabei den Pointer auf einen einzelnen Teilnehmer zurückgeben.
3. Implementieren Sie nun die Funktion `Teilnehmer** readAll(int* num)`. Diese Funktion soll die ganze Datei „input.txt“ einlesen, die Anzahl an Teilnehmern in num speichert und ein Array von Pointern auf Teilnehmer zurückgibt.
4. Schreiben sie eine Funktion
   `Teilnehmer* getTeilnehmer(Teilnehmer** liste, int numTeilnehmer, int platzierung)`, die aus der Liste einen beliebigen Teilnehmer mit einer bestimmten Platzierung (Funktionsargument) zurückgibt. Falls es keinen Teilnehmer mit der Platzierung gibt, soll sie NULL zurückgeben. Vergessen Sie nicht, dass Sie die Anzahl an Teilnehmern als Funktionsargument benötigen.
5. Implementieren sie nun eine Funktion
   `void freeTeilnehmer(Teilnehmer** liste, int num\_teilnehmer)`, die den Speicherplatz für die Liste an Teilnehmern freigibt.
6. *Challenge*: Überprüfen Sie, ob Platzierungen der Teilnehmer korrekt sind. Gehen Sie nach folgenden Regeln vor: Falls es mehr als einen Teilnehmer mit einer Platzierung gibt, werden die nächsten Plätze nicht vergeben. Dass heißt, falls es 3 zweite Plätze gibt, bleiben Platz vier und fünf auch frei usw. Mit dieser Ausnahme muss sonst jeder Platz mindestens einmal vergeben sein, d.h. auch es gibt immer mindestens einen ersten Platz. Die Funktion soll 1 zurückgeben, falls die Platzierungen korrekt sind, sonst 0.

Lösung

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    char vorname[64];
    char nachname[64];
    char geschlecht;
    int platzierung;
} Teilnehmer;

Teilnehmer* readTeilnehmer(FILE* fp) {
    char geschlecht;
    Teilnehmer* tn = malloc(sizeof(Teilnehmer));
    int i = fscanf(fp, "%63s␣%63s␣%c␣%d␣",
      tn->vorname,
      tn->nachname,
      &(tn->geschlecht),
      &(tn->platzierung)
    );
```

```
20      return tn;
21  }
22  Teilnehmer** readAll(int* num){
23    FILE* fp = fopen("input.txt", "r");
24    fscanf(fp, "%d␣", num);
25
26    Teilnehmer** liste = calloc(
27      sizeof(Teilnehmer*), (*num)
28    );
29
30    for(int i = 0; i < *num; i++){
31      Teilnehmer* t = readTeilnehmer(fp);
32      liste[i] = t;
33    }
34    return liste;
35  }
36
37  void freeTeilnehmer(Teilnehmer** liste, int numTeilnehmer) {
38    for(int i = 0; i < numTeilnehmer; i++) {
39      free(liste[i]);
40    }
41    free(liste);
42  }
43
44  Teilnehmer* getTeilnehmer(Teilnehmer** liste,
45    int numTeilnehmer, int platzierung){
46    for(int i = 0; i < numTeilnehmer; i++){
47      if(liste[i]->platzierung == platzierung)
48        return liste[i];
49    }
50    return NULL;
51  }
52
53  //Challenge
54  int verify(Teilnehmer** liste, int numTeilnehmer){
55    int current = 1;
56    while(current <= numTeilnehmer){
57      int found = 0;
58      for(int i = 0; i < numTeilnehmer; i++)
59        if(liste[i]->platzierung == current)
60          found++;
61      if(found == 0) return 0;
62      current += found;
63    }
64    return 1;
65  }
66
67  //Um das Programm auszuführen; Kein Teil der Aufgabenstellung
68  int main(int argc, char** argv) {
69    int num;
70    Teilnehmer** t = readAll(&num);
71    printf("%s\n", getTeilnehmer(t, num, 1)->vorname);
72    printf("%d\n", verify(t, num));
73    freeTeilnehmer(t, num);
74  }
75
76
77  //Alternativlösung eines anderes Tutor.
78  //Können Sie herausfinden, was das Programm tut (nicht klausurrelevant)?
79  //Dies ist ein Negativbeispiel für unleserlichen Code
80  #include <stdio.h>
```

```c
#include <stdlib.h>

typedef struct teilnehmer {
   char vorname[64];
   char nachname[64];
   char geschlecht;
   int plazierung;
} teilnehmer_t;

teilnehmer_t* readTeilnehmer(FILE* input){
    teilnehmer_t* t = malloc(sizeof(teilnehmer_t));
    return fscanf(input, "%63s %63s %c %d", t->vorname, t->nachname,
      &t->geschlecht, &t->plazierung), t;
}

teilnehmer_t** readAllTeilnehmer(FILE* input, int* anzahl){
    teilnehmer_t** t = calloc(sizeof(teilnehmer_t*),
      (fscanf(input, "%d", anzahl), *anzahl));
    for(int i = 0; i++ < *anzahl; *(t++) = readTeilnehmer(input));
    return t - *anzahl;
}

teilnehmer_t* getPlaziert(teilnehmer_t** teilnehmer, int anzahl, int platz){
    while(anzahl--)
        if(teilnehmer[anzahl]->plazierung == platz)
            return teilnehmer[anzahl];
    return 0;
}

int teilnehmerOk(teilnehmer_t** teilnehmer, int anzahl){
    for(int platz = 1, anzahl2 = anzahl, count = (anzahl = 1);
      platz <= anzahl2; platz += count){
        for(int i = (count = 0); i < anzahl2; i++)
            count += teilnehmer[i]->plazierung == platz;
        anzahl &= !!count;
    }
    return anzahl;
}

void destroyTeilnehmer(teilnehmer_t** teilnehmer, int anzahl){
    for(;anzahl--;free(teilnehmer[anzahl]));
    free(teilnehmer);
}

int main(int argc, char ** argv){
    FILE* input = fopen("input.txt", "r");
    int anzahl;
    teilnehmer_t** t = readAllTeilnehmer(input, &anzahl);
    fclose(input);
    printf("OK: %d\n", teilnehmerOk(t, anzahl));
    destroyTeilnehmer(t, anzahl);
    return 0;
}
```

Aufgabe 2: Malloc is Magic

Bestimmen Sie von Hand, welche Ausgabe das folgende Programm liefert. Benutzen Sie den Computer nur, um zu überprüfen, ob Ihr Ergebnis korrekt ist.

```c
    #include <stdio.h>
```

```
2     #include <stdlib.h>
3     int main(int argc, char* argv[]) {
4       int *buf = calloc(sizeof(int), 4);
5
6       int i = *buf;
7       int* ip = &i;
8       int* ip2 = buf+2;
9       int** ipp = &ip;
10
11      for (int k = 0; k < 4; k++) {
12        if (k % 2 == 0) {
13          *ip = **ipp + 1;
14          *ip2 = buf - ip2;
15          ip2++;
16        } else {
17          *ipp = buf + i;
18          ipp = &ip2;
19          *ip = 42;
20        }
21      }
22      printf("%d␣%d␣%d␣%d␣%d␣%td\n", buf[0], buf[1], buf[2], buf[3],
23             i, buf - *ipp);
24      free(buf);
25      return 0;
26    }
```

Lösung

Ausgabe des Programms: 0 42 -2 -3 1 -1

Schritt	Zeile	buf	buf[0]	buf[1]	buf[2]	buf[3]	i	ip	ip2	ipp	k
1	4	&buf[0]	0	0	0	0	-	-	-	-	-
2	6	&buf[0]	0	0	0	0	0	-	-	-	-
3	7	&buf[0]	0	0	0	0	0	&i	-	-	-
4	8	&buf[0]	0	0	0	0	0	&i	&buf[2]	-	-
5	9	&buf[0]	0	0	0	0	0	&i	&buf[2]	&ip	-
6	11	&buf[0]	0	0	0	0	0	&i	&buf[2]	&ip	0
7	13	&buf[0]	0	0	0	0	1	&i	&buf[2]	&ip	0
8	14	&buf[0]	0	0	-2	0	1	&i	&buf[2]	&ip	0
9	15	&buf[0]	0	0	-2	0	1	&i	&buf[3]	&ip	0
10	11	&buf[0]	0	0	-2	0	1	&i	&buf[3]	&ip	1
11	17	&buf[0]	0	0	-2	0	1	&buf[1]	&buf[3]	&ip	1
12	18	&buf[0]	0	0	-2	0	1	&buf[1]	&buf[3]	&ip2	1
13	19	&buf[0]	0	42	-2	0	1	&buf[1]	&buf[3]	&ip2	1
14	11	&buf[0]	0	42	-2	0	1	&buf[1]	&buf[3]	&ip2	2
15	13	&buf[0]	0	-1	-2	0	1	&buf[1]	&buf[3]	&ip2	2
16	14	&buf[0]	0	-1	-2	-3	1	&buf[1]	&buf[3]	&ip2	2
17	15	&buf[0]	0	-1	-2	-3	1	&buf[1]	&buf[4]	&ip2	2
18	11	&buf[0]	0	-1	-2	-3	1	&buf[1]	&buf[4]	&ip2	3
19	17	&buf[0]	0	-1	-2	-3	1	&buf[1]	&buf[1]	&ip2	3
20	18	&buf[0]	0	-1	-2	-3	1	&buf[1]	&buf[1]	&ip2	3
21	19	&buf[0]	0	42	-2	-3	1	&buf[1]	&buf[1]	&ip2	3
22	11	&buf[0]	0	42	-2	-3	1	&buf[1]	&buf[1]	&ip2	4
23	21	&buf[0]	0	42	-2	-3	1	&buf[1]	&buf[1]	&ip2	-
24	24	-	-	-	-	-	1	-	-	&ip2	-

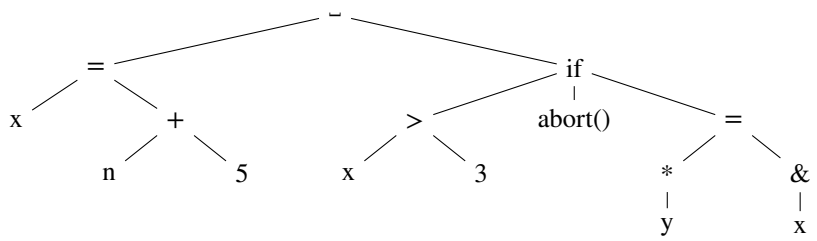
Aufgabe 3: C0 AST

Zeichnen Sie den abstrakten Syntaxbaum des folgenden C0-Programms:

```
x = n + 5;
if (x > 3)
    abort();
else
    *y = &x;
```

Lösung

Auf der „obersten Ebene" hat das Programm eine Hintereinanderausführung von `x = n + 5` und dem if-Statement. Daher ist die Wurzel des ASTs eine Hintereinanderausführung ␣ und die Kinder sind die Zuweisung und das if-Statement. Bei einem binären Operator ist der entsprechende Operator ein Knoten und die linke und rechte Seite ist jeweils ein Kind davon. Unäre Operatoren haben ihr Argument als Kind. Beachten Sie, dass insbesondere * und & auch unäre Operatoren sind. Das if-Statement hat drei Kinder: die Bedingung, den Fall, wenn die Bedingung wahr ist und den Fall, wenn die Bedingung falsch ist. Damit ergibt sich folgender Baum:



Aufgabe 4: C0pb Ausführung

Führen Sie folgende Programme mit der Semantik von C0pb auf den jeweils angegebenen Zuständen aus. Geben Sie auSSerdem an, ob das Programm terminiert, abbricht oder stecken bleibt.

1. ```
   a = 12;
   b = a + 18;
   c = &b;
   *c = *c + a;
   ```

   $$\sigma = \{\mathtt{a} \mapsto \triangle, \mathtt{b} \mapsto \pentagon, \mathtt{c} \mapsto \diamond\}; \{\triangle \mapsto ?, \pentagon \mapsto ?, \diamond \mapsto ?\}$$

   Geben Sie zusätzlich die Auswertung von $[\![\mathtt{\&b}]\!]$ und $[\![\mathtt{*c + a}]\!]$ explizit an.

2. ```
   {
       int x;
       x = 7;
       if (x > 3)
           abort();
       else
           x = 0;
   }
   ```

   $$\sigma = \{\}; \{\}$$

3.
```
x = 5;
y = 3;
{
    int y;
    y = x + 7;
}
```

$$\sigma = \{\texttt{x} \mapsto \triangle\}; \{\triangle \mapsto ?\}$$

Lösung

1.

$$\begin{aligned}
&\langle \texttt{a = 12; b = a + 18; c = \&b; *c = *c + a; } \mid \\
&\quad \{\texttt{a} \mapsto \triangle, \texttt{b} \mapsto \pentagon, \texttt{c} \mapsto \diamondsuit\}; \{\triangle \mapsto ?, \pentagon \mapsto ?, \diamondsuit \mapsto ?\}\rangle \\
\rightarrow\ &\langle \texttt{b = a + 18; c = \&b; *c = *c + a; } \mid \\
&\quad \{\texttt{a} \mapsto \triangle, \texttt{b} \mapsto \pentagon, \texttt{c} \mapsto \diamondsuit\}; \{\triangle \mapsto 12, \pentagon \mapsto ?, \diamondsuit \mapsto ?\}\rangle && \text{[Assign], [Exec]} \\
\rightarrow\ &\langle \texttt{c = \&b; *c = *c + a; } \mid \\
&\quad \{\texttt{a} \mapsto \triangle, \texttt{b} \mapsto \pentagon, \texttt{c} \mapsto \diamondsuit\}; \{\triangle \mapsto 12, \pentagon \mapsto 30, \diamondsuit \mapsto ?\}\rangle && \text{[Assign], [Exec]} \\
\rightarrow\ &\langle \texttt{*c = *c + a; } \mid \\
&\quad \{\texttt{a} \mapsto \triangle, \texttt{b} \mapsto \pentagon, \texttt{c} \mapsto \diamondsuit\}; \{\triangle \mapsto 12, \pentagon \mapsto 30, \diamondsuit \mapsto \pentagon\}\rangle && \text{[Assign], [Exec]} \\
\rightarrow\ &\{\texttt{a} \mapsto \triangle, \texttt{b} \mapsto \pentagon, \texttt{c} \mapsto \diamondsuit\}; \{\triangle \mapsto 12, \pentagon \mapsto 42, \diamondsuit \mapsto \pentagon\} && \text{[Assign]}
\end{aligned}$$

$$\begin{aligned}
[\![\texttt{\&b}]\!]\sigma &= [\![\texttt{b}]\!]_L\sigma \\
&= \rho(b) \\
&= \pentagon
\end{aligned}$$

$$\begin{aligned}
[\![\texttt{*c + a}]\!]\sigma &= [\![\texttt{*c}]\!]\sigma + [\![\texttt{a}]\!]\sigma \\
&= \mu[\![\texttt{*c}]\!]_L\sigma + \mu[\![\texttt{a}]\!]_L\sigma \\
&= \mu[\![\texttt{c}]\!]\sigma + \mu(\rho(a)) \\
&= \mu(\mu([\![\texttt{c}]\!]_L\sigma)) + \mu(\triangle) \\
&= \mu(\mu(\rho(c))) + 12 \\
&= \mu(\mu(\diamondsuit)) + 12 \\
&= \mu(\pentagon) + 12 \\
&= 30 + 12 \\
&= 42
\end{aligned}$$

Das Programm terminiert im Zustand

$$\{\texttt{a} \mapsto \triangle, \texttt{b} \mapsto \pentagon, \texttt{c} \mapsto \diamondsuit\}; \{\triangle \mapsto 12, \pentagon \mapsto 42, \diamondsuit \mapsto \pentagon\}$$

2.

$$\begin{aligned}
&\langle \{\texttt{int x; x = 7; if (x > 3) abort(); else x = 0;}\} \mid \{\}; \{\}\rangle \\
\rightarrow\ &\langle \texttt{x = 7; if (x > 3) abort(); else x = 0;}\blacksquare \mid \{\}, \{\texttt{x} \mapsto \triangle\}; \{\triangle \mapsto ?\}\rangle && \text{[Block]} \\
\rightarrow\ &\langle \texttt{if (x > 3) abort(); else x = 0;}\blacksquare \mid \{\}, \{\texttt{x} \mapsto \triangle\}; \{\triangle \mapsto 7\}\rangle && \text{[Assign], [Exec]} \\
\rightarrow\ &\langle \texttt{abort();}\blacksquare \mid \{\}, \{\texttt{x} \mapsto \triangle\}; \{\triangle \mapsto 7\}\rangle && \text{[IfTrue], [Exec]} \\
\rightarrow\ &\lightning && \text{[Abort], [Crash]}
\end{aligned}$$

Das Programm bricht ab.

3.

$$\begin{aligned}
&\langle \texttt{x = 5; y = 3; \{int y; y = x + 7\}} \mid \{\texttt{x} \mapsto \triangle\}; \{\triangle \mapsto ?\}\rangle \\
\rightarrow\ &\langle \texttt{y = 3; \{int y; y = x + 7\}} \mid \{\texttt{x} \mapsto \triangle\}; \{\triangle \mapsto 5\}\rangle && \text{[Assign], [Exec]}
\end{aligned}$$

Das Programm bleibt stecken, da y in der Zuweisung `y = 3` undefiniert ist (y ist nicht in ρ enthalten).

Aufgabe 5: Zweige & Pfade

Betrachten Sie folgendes Programm:

```
1 int check (int x){
2     if (x < 0) {
3         return -1;
4     }
5     int res = 0;
6     if (x % 4 == 0) {
7         res = 1;
8     }
9     if (x % 8 == 0) {
10        res = 2;
11    }
12    return res;
13 }
```

1. Schreiben Sie eine Testsuite, die alle Anweisungen abdeckt.
2. Erweitern Sie Ihre Testsuite so, sodass nun auch alle Zweige abgedeckt werden.
3. Deckt Ihre Testsuite alle Pfade ab? Begründen sie wieso (nicht)!
4. Machen Sie sich den Unterschied zwischen Blackbox/Whitebox und Funktionsbasiert/Spezifikationsbasiert bewusst. Können Tests aussagen über die Korrektheit eines Programms machen?

Lösung

1.

```
1 void test_negative() {
2     int res = check(-1);
3     assert(res == -1);
4 }
5
6 void test_both() {
7     int res = check(0);
8     assert(res == 2);
9 }
```

2.

```
1 void test_one() {
2     int res = check(1);
3     assert(res == 0);
4 }
```

3. Nein! Es gibt noch 2 unabgedeckte Pfade: einmal der Fall, dass x durch 4 teilbar, aber nicht durch 8 teilbar ist und einmal der Fall, dass x durch 8 aber nicht durch 4 teilbar ist. Letzterer Fall ist allerdings unmöglich, und der Pfad ist daher kein möglicher Pfad und kann ignoriert werden. Um Pfadabdeckung zu erreichen müsste also nurnoch ein Fall wie bspw. $x = 4$ eingefügt werden.
4. Tests zeigen nur, dass ein Programm für den gegebenen Input nicht kaputt ist, nicht, dass das Programm allgemein korrekt ist. Wenn die Unterschiede nicht klar sind vgl. mit dem Skript.

Aufgabe 6: Mapple Mypods

Die Firma Mapple produziert seit einigen Jahren eine MP3-Player Reihe namens MyPod. Bisher sind die Modelle Mykro, Mega und X erschienen. Im folgenden sollen Sie die Funktionalitäten der MP3-Player in Java modellieren. Erstellen Sie dazu je eine Klasse pro Modell und stellen Sie folgende Funktionalitäten bereit:
(Lösungsansatz): Dadurch, dass eine Klasse pro Modell erstellt werden muss, weiSS man dass es mindestens die Klassen Mykro, Mega und X geben soll. Da die Modelle (wie man im weiteren Text feststellt) geteilte Funktionalitäten bereit stellen sollen, sollten die Klassen alle von einer gemeinsamen Klasse (MyPod) erben. Hierbei sind die Namen eigentlich beliebig wählbar, die von uns gewählten Namen sind aber aus dem Text sehr naheliegend.

- Jeder MP3-Player hat einen Modellnamen und eine Seriennummer.
  (Lösungsansatz): Damit müssen in der Klasse MyPod jeweils für Modellname (hier: modelName) und Seriennummer (hier: serialNumber) ein Attribut angelegt werden. Es ist wichtig diese Attribute und die damit verbundene Funktionalität in MyPod anzulegen, damit man keinen Code dupliziert. Da die Attribute Nummern sind können wir sie als Ganzzahl (int) darstellen.
  Der Modellname ist eindeutig durch das Modell festgelegt (Mykro, Mega oder X).
  (Lösungsansatz): Hierdurch weiSS man, dass dieser Wert als Strinkonstante (eindeutig durch die Klasse bestimmt) in den jeweiligen Unterklassen an den Konstruktor der Oberklasse übergeben werden kann.
  Die Seriennummer muss initial für jeden MP3-Player individuell festgelegt werden.
  (Lösungsansatz): Hierdurch weiSS man, dass dieser Seriennummer an den Konstruktor der jeweiligen Unterklassen übergeben werden muss. (Um diesen Wert dann an die Oberklasse weiterzugeben.)
  Beide Eigenschaften sollen von auSSen erfragt werden können, aber dürfen nicht veränderbar sein.
  (Lösungsansatz): Da die Eigenschaften nicht verändert werden dürfen, setzen wir sie als final. (Einmalig im Konstruktor beschrieben.) Da die von auSSen erfragbar sein sollen, erstellen wir für beide Attribute einen (public) Getter.

- Des weiteren soll die Anzahl der Akkuladungen festgehalten werden.
  (Lösungsansatz): Damit benötigt man ein weiteres Attribut (hier: chargeAmount). Da das Attribut eine Anzahl ist wollen wir es als Ganzzahl (int) darstellen. Da die Funktionalität von allen MP3-Playern geteilt wird, soll dieses Attribut in MyPod liegen.
  Diese ist zu Anfang immer 0 und soll durch eine Methode charge um 1 inkrementiert werden können.
  (Lösungsansatz): Da Attribute einer Klasse automatisch mit 0 initialisiert werden, muss man chargeAmount nicht explizit einen Wert zuweisen. Da die Methode charge keine weiteren Informationen benötigt und nicht zurück gibt hat sie die Signatur void charge().
  Die Anzahl der Ladungen soll auf keinen anderen weg von auSSen veränderbar sein, aber aber erfragbar abfragbar.
  (Lösungsansatz): Damit wird ein (public) Getter für chargeAmount benötigt.

- Jeder MP3-Player merkt sich den Namen des momentan wiedergegeben Liedes.
  (Lösungsansatz): Damit benötigt man ein weiteres Attribut (hier: currentSong). Da dieses Attribut ein Name ist, wollen wir es als Zeichenkette (String) darstellen. Da das Attribut von allen MP3-Playern benötigt wird, soll dieses Attribut in MyPod liegen.
  Dieser wird auf „No Song“ gesetzt, falls derzeit kein Lied wiedergegeben wird.
  (Lösungsansatz): Damit wird currentSong mit „No Song“ initialisiert. Manchmal lohnt es sich wenn man eine String Konstante öfter benutzt, dafür eine Konstante (hier: `NO_SONG`) anzulegen, damit man diese im Code benutzen kann. Wir haben uns hier dafür entschieden. (Das ist aber kein muss.)
  Dies ist zu Beginn der Fall und kann durch eine Methode songStoppedPlaying angezeigt werden.
  (Lösungsansatz): Da die Methode songStoppedPlaying keine weiteren Argumente benötigt und nicht zurück gibt is die Signatur void songStoppedPlaying().
  Der Name des momentan wiedergegeben Liedes wieder soll erfragbar sein und durch eine Methode play gesetzt werden können.
  (Lösungsansatz): Damit das Attribut erfragbar ist, wird ein (public) Getter in MyPod für currentSong eingefügt. Die Methode play benötigt den Namen des Liedes und gibt zurück ob etwas funktioniert hat (ja/nein). Dieses ja/nein kann man als wahr oder falsch also einen Wahrheitswert (boolean) interpretieren. Damit is die Signatur von play: boolean play(String song).
  Die Methode soll zurückgeben, ob der momentane Song abgespielt werden kann. Beim Modell Mykro und Mega kann immer ein beliebiger Song wiedergegeben werden. Beim Modell X soll das abspielen nur Möglich sein wenn genug Guthaben aufgeladen wurde.
  (Lösungsansatz): Deswegen implementieren wir die Grundlegende Funktionalität in MyPod und ergänzen die erweiterte Funktionalität in X.

- Guthaben wird nur auf dem Modell X verwaltet und funktioniert wie folgt:
  (Lösungsansatz): Damit benötigt man ein weiteres Attribut (hier: balance). Da wir immer in 10 Cent schritten Zählen, und nur Ganzzahlige Beträge hinzugefügt werden, müssen wir dieses Guthaben nicht unbedingt als Kommazahl darstellen. WIr haben uns hier entschieden einfach in 10 Cent schritten zu zählen. Z.B. entspricht der Wert 36 in Balance dann 360 Cent und somit 3,60. Somit können wir es auch als Integer (Ganzzahl) darstellen. Die Lösung mit float wäre aber auch valide gewesen.
  - Das Abspielen eines Liedes kostet 10 Cent.
  - Das Guthaben auf einem neuen MP3-Player des Modells X beträgt 15.
  - Es kann mehr Guthaben durch eine Methode addBalance hinzugefügt werden. Dabei können immer nur volle Euro-Beträge aufgeladen werden.
    (Lösungsansatz): Da wir einen ganzzahligen Wert (int) übergeben bekommen und nichts zurückgeben müssen ist die Signatur von addBalance also void addBalance(int amount). Da wir in 10Cent schritten Zählen muss der Euro-Betrag noch mit 10 multipliziert werden.
- Des weiteren verfügt jeder MP3-Player über eine Check-Sum die über eine Methode Namens checkSum berechnet werden soll. Diese Berechnung weicht von Modell zu Modell ab und funktioniert wie folgt:
  (Lösungsansatz): Da jeder MP3-Player eine Check-Sum hat, wir diese aber immer unterschiedlich berechnen, wird sie jeweils unteschiedlich in den Unterklassen implementiert. (Und als abstrakte Funktion in MyPod gelistet.) Da die Funktion nichts übergeben bekommt und eine Ganzzahl (int) zurückgeben soll ist die Signatur int checkSum().
  - Bei Mykro wird die Quersumme der Seriennummer berechnet.
  - Bei Mega wird Seriennummer modulo 13 genommen.
  - Bei X wird die Seriennummer durch 7 geteilt.

1. Implementieren Sie die oben beschriebenen Klassen und Funktionalitäten. Ihnen steht es frei beliebige weitere Klassen zu erstellen. Es muss darauf geachtet werden Code-Duplizierung zu vermeiden und übliche Java-Standards einzuhalten, die Sie in der Vorlesung kennengelernt haben.
2. Implementieren sie eine Funktion checkSumEqual. Diese bekommt einen MP3-Player und eine Ganzzahl übergeben und soll überprüfen ob die Check-Sum des MP3-Players mit der übergebenen Zahl übereinstimmt.
3. Schreiben Sie ein Stück Code, das einen MP3-Player des Modells Mega mit der Seriennummer 123 erstellt und den Song „Yeah, Heavysaurus!“ wiedergibt.

Lösung

1. Im Folgenden finden Sie eine Beispielimplementierung. Das Meiste ist n der Aufgabenstellung wurde an die jeweilige Textstelle annotiert welcher Code wieso aus ihr entstanden ist. Hier noch ein paar allgemeine Hinweise zum Design:
   - Solange es um Attribute einer Klasse geht sollten die Sichtbarkeitsmodifizierer für das Attribut private oder protected sein.
     - private, wenn nur die eigentliche Klasse auf das Attribut zugreifen können soll.
     - protected, wenn auch die Unterklassen das Attribut benutzen sollen.
   - Wie oben angemerkt sollten Attribute nicht public sein, falls ein Attribut also von auSSen verändert werden können sollen braucht man einen Setter. Falls sie von auSSen gelsen werden sollen, braucht man einen Getter. Die meisten Getter/Setter können auf folgende Weise generisch implementiert werden:

```
1 public void setAttr(T attr) {
2     this.attr = attr;
3 }
```

```
1 public T gettAttr() {
2     return attr;
3 }
```

Ab hier beginnt die Beispielimplementierung:

```
1 public abstract class MyPod {
2     private final static String NO_SONG = "No Song";
3     private final String modelName;
4     private int chargeAmount;
5     protected final int serialNumber;
6     private String currentSong;
7
8     protected MyPod(String modelName, int serialNumber) {
9         this.modelName = modelName;
10        this.serialNumber = serialNumber;
11        currentSong = NO_SONG;
12    }
13
14    public String getModelName() {
15        return modelName;
16    }
17
18    public int getChargeAmount() {
19        return chargeAmount;
20    }
21
22    public int getSerialNumber() {
23        return getSerialNumber();
24    }
25
26    public void charge() {
27        chargeAmount++;
28    }
29
30    public boolean play(String song) {
31        currentSong = song;
32        return true;
33    }
34
35    public void stoppedPlaying() {
36        currentSong = NO_SONG;
37    }
38
39    public abstract int checkSum();
40 }
```

```
1 public class Mykro extends MyPod {
2
3     public Mykro(int serialNumber) {
4         super("Mykro", serialNumber);
5     }
6
7     @Override
8     public int checkSum() {
9         int num = serialNumber;
10        int sum = 0;
11        while (num > 0) {
12            sum = sum + num % 10;
13            num = num / 10;
14        }
15        return num;
16    }
17 }
```

```java
public class Mega extends MyPod {

    public Mega(int serialNumber) {
        super("Mega", serialNumber);
    }

    @Override
    public int checkSum() {
        return serialNumber % 13;
    }
}
```

```java
public class X extends MyPod {
    int balance;

    public X(int serialNumber) {
        super("X", serialNumber);
        balance = 150;
    }

    @Override
    public int checkSum() {
        return serialNumber / 7;
    }

    public void addBalance(int amount) {
        balance += amount * 10;
    }

    @Override
    public boolean play(String song) {
        if (balance == 0)
            return false;
        balance--;
        return super.play(song);
    }
}
```

2. Hat man in der Aufgabe zuvor checkSum als abstrakte Methode implementiert ist die Lösung hier relativ simpel:

```java
public boolean checkSumEqual(MyPod myPod, int compareTo) {
    return myPod.checkSum() == compareTo;
}
```

   Man muss lediglich die Check-Sums vergleichen und das Ergebnis zurück geben. == funktioniert hier, da es sich um Ganzzahlen (ints) handelt.

3. Die Seriennummer muss an den Konstruktor von Mega übergeben werden, der Songname beim Aufruf von play auf dem erstellten Objekt. Die Lösung sieht somit wie folgt aus:

```java
Mega myPod = new Mega(123);
myPod.play("Yeah, Heavysaurus!");
```

Aufgabe 7: Vererbungshierarchie

Welche Methode wird jeweils aufgerufen? Geben Sie die Ausgabe des Programms an.

Klassenhierarchie

```java
public class A {
  public void fun(B b) {
    System.out.println("A.fun(B)");
  }
  public void fun(D d) {
    System.out.println("A.fun(D)");
  }
}
```

```java
public class B extends A {
  public void fun(A a) {
    System.out.println("B.fun(A)");
  }
  public void fun(C c) {
    System.out.println("B.fun(C)");
  }
  public void fun(E e) {
    System.out.println("B.fun(E)");
  }
}
```

```java
public class C extends B {
  public void fun(B b) {
    System.out.println("C.fun(B)");
  }
}
```

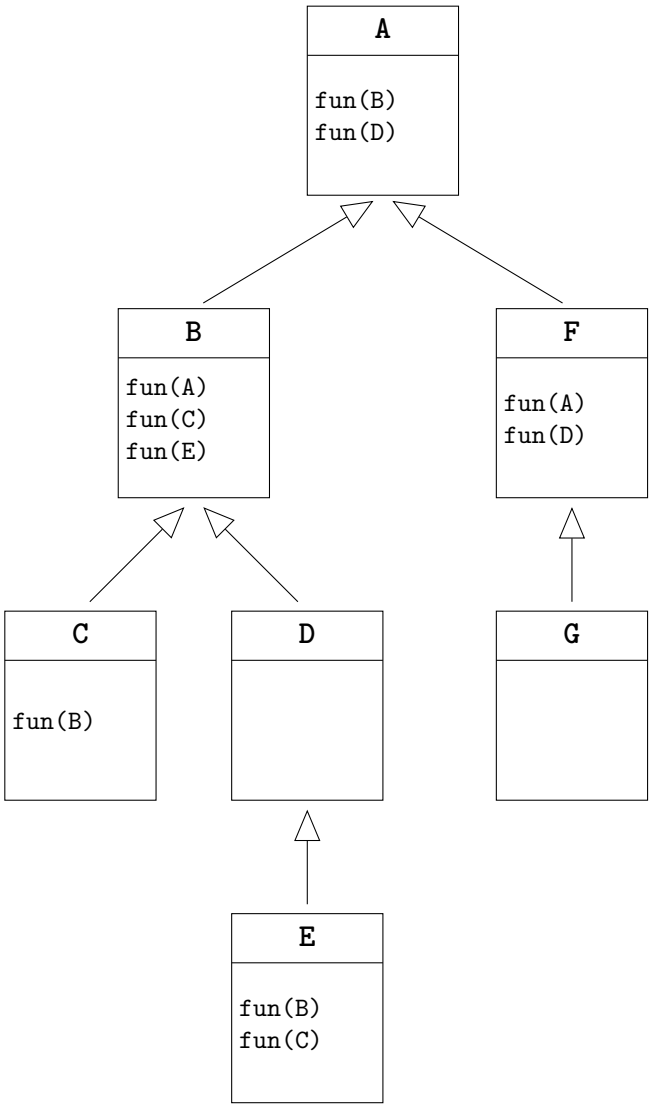
```java
public class D extends B {
}
```

```java
public class E extends D {
  public void fun(B b) {
    System.out.println("E.fun(B)");
  }
  public void fun(C c) {
    System.out.println("E.fun(C)");
  }
}
```

```java
public class F extends A {
  public void fun(A a) {
    System.out.println("F.fun(A)");
  }
  public void fun(D d) {
    System.out.println("F.fun(D)");
  }
}
```

```java
public class G extends F {
}
```

```java
public class Main {
    public static void main(
        String[] args) {
        A aa = new A();
        A ab = new B();
        A ac = new C();
        A ag = new G();
        A af = new F();
        B bb = new B();
        B bc = new C();
        B bd = new D();
        B be = new E();
        C cc = new C();
        D dd = new D();
        D de = new E();
        E ee = new E();
        F ff = new F();
        F fg = new G();
        G gg = new G();

        aa.fun(bd);
        ac.fun(bb);
        ac.fun(be);
        af.fun(dd);
        af.fun(ab);
        ag.fun(cc);

        bb.fun(ab);
        bb.fun(bc);
        bc.fun(bb);
        bc.fun(ff);
        bc.fun(dd);
        bd.fun(cc);
        be.fun(bc);
        be.fun(cc);
        be.fun(dd);

        cc.fun(de);
        cc.fun(ee);

        dd.fun(dd);
        dd.fun(bd);
        dd.fun(fg);

        ee.fun(ee);
        ee.fun(cc);

        ff.fun(fg);
        gg.fun(de);
        gg.fun(bc);
    }
}
```



Lösung

Die Lösungen können nach dem folgenden Schema berechnet werden:
Damit ergibt sich:

```
aa.fun(bd); => A.fun(B)
ac.fun(bb); => C.fun(B)
ac.fun(be); => C.fun(B)
```

```
af.fun(dd); => F.fun(D)
af.fun(ab); => Fehler!
ag.fun(cc); => A.fun(B)

bb.fun(ab); => B.fun(A)
bb.fun(bc); => A.fun(B)
bc.fun(bb); => C.fun(B)
bc.fun(ff); => B.fun(A)
bc.fun(dd); => A.fun(D)
bd.fun(cc); => B.fun(C)
be.fun(bc); => E.fun(B)
be.fun(cc); => E.fun(C)
be.fun(dd); => A.fun(D)

cc.fun(de); => A.fun(D)
cc.fun(ee); => B.fun(E)

dd.fun(dd); => A.fun(D)
dd.fun(bd); => A.fun(B)
dd.fun(fg); => B.fun(A)

ee.fun(ee); => B.fun(E)
ee.fun(cc); => E.fun(C)

ff.fun(fg); => F.fun(A)
gg.fun(de); => F.fun(D)
gg.fun(bc); => A.fun(B)
```

Aufgabe 8: Wer hasht was?

1. Was ist der Vorteil von Datenstrukturen die mithilfe von Hashing den Zugriff auf die Elemente realisiert?
2. Was ist der Zusammenhang zwischen *HashCode* und *Equals*?
3. Auf was ist bei der Implementierung von Hash-Funktionen zu achten?
4. Welche grundlegenden Techniken zum Implementieren von Hash-Tabellen sind Ihnen bekannt? Wo liegen die Vorteile/Nachteile?

Lösung

1. Meistens ist der Zugriff auf die Elemente wesentlich schneller als bei der Verwendung von z.B Arrays oder Listen (beispielsweise wenn ein bestimmtes Objekt in einer Datenstruktur gesucht wird).
2. $\forall\, p, q : p.equals(q) \Rightarrow p.hashCode() == q.hashCode()$.
3. Die gewählte Hash-Funktion sollte eine möglichst gute Verteilung erzeugen (im Idealfall ist sie injektiv).
4. Kollisionslisten, lineares Sondieren und quadratisches Sondieren.

   Kollisionslisten haben den Nachteil, dass für Listeneinträge zusätzlicher Speicher angefordert werden muss, vorallem wenn die Tabelle vergrößert werden soll und alle Listeneinträge zerstört und neu aufgebaut werden müssen.

   Beim linearen Sondieren ist zwar garantiert, dass allen Zellen der Hashtabelle ein Wert zugeordnet werden kann, allerdings bilden sich hierbei schnell Anhäufungen von Daten, was die Leistungsfähigkeit unter Umständen stark beeinträchtigen kann.

Beim quadratischen Sondieren hingegen soll die Bildung solcher Häufungen vermindert werden. Hier ist jedoch das Problem, dass die gewählte Hashfunktion surjektiv auf die Menge der Tabelleneinträge abbildet.

Aufgabe 9: Fehlersuche

Was ist die Ausgabe des folgenden Programms? Was ist das Problem?

```
1  public class HT {
2      int a;
3
4      public HT(int a) {
5          this.a = a;
6      }
7
8      public HT setA(int a) {
9          this.a = a;
10         return this;
11     }
12
13     @Override
14     public int hashCode() {
15         return a;
16     }
17
18     @Override
19     public boolean equals(Object o) {
20         if (!(o instanceof HT))
21             return false;
22         HT ht = (HT) o;
23         return a == ht.a;
24     }
25
26     public static void main(String[] args) {
27         HT ht = new HT(10);
28         HashSet<HT> set = new HashSet<>(100);
29         set.add(ht);
30         System.out.println(set.contains(ht));
31         ht.setA(20);
32         System.out.println(set.contains(ht));
33     }
34 }
```

Lösung

Das Programm gibt `true` und `false` aus!
Durch `setA(int a)` wird der Wert `a` in `ht` verändert. Somit verändert sich der Hash von `ht` und es kann unter dem ursprünglichen Key nicht mehr gefunden werden.

Aufgabe 10: Kollisionslisten

Konrad Klug beobachtet in seiner Freizeit oft die Fische im Walgenbach, der durch das Forstgebiet Oswald flieSSt. Er ist so fasziniert von ihnen, dass er beschlossen hat, ein Programm über sie zu schreiben. Er hat bereits eine Klasse `Fisch` angelegt. Nun will er die Fische in eine Streutabelle einordnen.

```
1      public class Fisch {
2          private int anzahlSchuppen;
3          private final String name;
4
5          public Fisch(String name, int anzahlSchuppen) {
6              this.anzahlSchuppen = anzahlSchuppen;
```

```
 7            this.name = name;
 8        }
 9
10        public void verliertSchuppe() {
11            anzahlSchuppen--;
12        }
13
14        public int hashCode() {
15            return anzahlSchuppen/100;
16        }
17
18        public boolean equals(Object o) {
19            if(!(o instanceof Fisch))
20                return false;
21
22            return ((Fisch) o).name.equals(name);
23        }
24    }
```

1. Um sich das Ganze besser vorstellen zu können, möchte Konrad Klug das Einhashen zuerst mit Stift und Papier nachvollziehen. Tun Sie das auch und hashen Sie die Fische aus folgender Tabelle in ein `HashSet` der GröSSe 7 ein. Sie können annehmen, dass die Streutabelle ihre Länge *nicht* verändert. Verwenden Sie zur Kollisionsauflösung Kollisionslisten.

Name	Anzahl der Schuppen
Bernd	112
Heidi	564
Margret	312
Karl	273
Heinz	532
Günther	750
Helene	37

2. Nachdem Konrad sich nun beim Einhashen sicher fühlt, schreibt er das folgende Programm zum initialen Test seiner Idee:

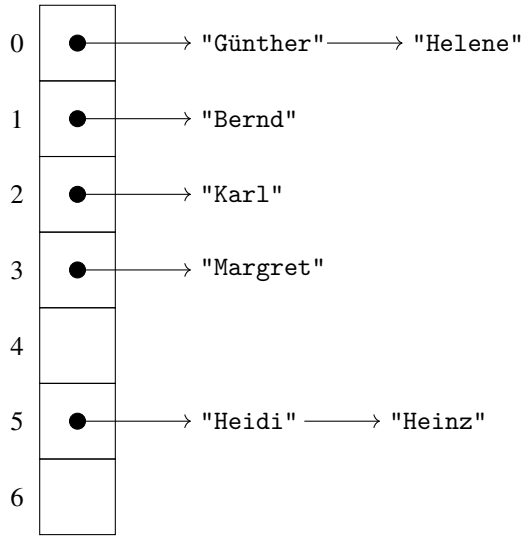
```
 1    public void test() {
 2        Fisch testFischEins = new Fisch("Test", 100);
 3        Fisch testFischZwei = new Fisch("Test", 200);
 4        Fisch testFischDrei = new Fisch("Test", 300);
 5        Set<Fisch> bach = new HashSet<>();
 6
 7        bach.add(testFischEins);
 8        bach.add(testFischZwei);
 9        bach.add(testFischDrei);
10        testFischEins.verliertSchuppe();
11        testFischDrei.verliertSchuppe();
12
13        if(bach.contains(testFischEins))
14            System.out.println("Fisch␣1␣schwimmt␣im␣Bach.");
15
16        if(bach.contains(testFischDrei))
17            System.out.println("Fisch␣3␣schwimmt␣im␣Bach.");
18    }
```

Welche Ausgabe würden Sie bei einer idealen Implementierung von `Fisch` erwarten? Was ist die Ausgabe mit Konrad's oben gegebener Implementierung von `Fisch`?

3. Weil Konrad es einfach nicht geschafft hat seinen Code zu reparieren ist No Hau zur Hilfe geeilt. Nach einem Hieb von No Hau's Codemagie macht die in (b) implementierte Methode nun auch was sie soll. Können Sie den Code von Klasse `Fisch` auch wie No Hau reparieren?

Lösung

1.



2. Da wir beide Fische in das Hashset eingefügt haben, erwarten wir bei einer korrekten Implementierung von `Fisch` die folgende Ausgabe:

```
1 Fisch 1 schwimmt im Bach.
2 Fisch 3 schwimmt im Bach.
```

   Bei der gegebenen Implementierung wird jedoch nur `"Fisch 3 schwimmt im Bach."` ausgegeben. Das liegt an den Aufrufen von `verliertSchuppe`. Denn dadurch ändern sich die HashCodes von `testFischEins` auf 0 (von 1) und von `testFischDrei` auf 2 (von 1).

   An Stelle 0 des Hashsets befindet sich nichts, weil `testFischEins` mit HashCode 1 eingefügt wurde. Also gibt `bach.contains(testFischEins)` false zurück.

   An Stelle 2 des Hashsets befindet sich `testFischZwei`. Es gilt `testFischZwei.equals(testFischDrei)` (weil ihre Namen gleich sind) also gibt `bach.contains(testFischDrei)` true zurück.

3. Wie wir gerade eben gesehen haben liegt das Problem daran, dass der HashCode sich im Programmverlauf ändern kann. Eine Idee das zu lösen (indem man `hashCode()` ändert) wäre die folgende:

```
1     public int hashCode() {
2         return name.hashCode();
3     }
```

   Da `name` als `final` deklariert wurde, kann sich dieser im Programmverlauf nicht mehr ändern und die Problematik nicht mehr auftreten.

Aufgabe 11: Lineares und quadratisches Sondieren

Betrachten Sie die Hashfunktion

$$h(x) = x \,\%\, m$$

wobei m die GröSSe der Hashtabelle ist. Nehmen Sie an, dass $m = 8$ ist. Fügen Sie nun die Elemente

```
8 3 11 0 16
```

in die Tabelle ein.

1. Nutzen Sie zum Auflösen von Konflikten lineares Sondieren. Notieren Sie die Ergebnisse der Streufunktion für die einzufügenden Elemente und ggf. die Ergebnisse der Sondierungsfunktion.
2. Nutzen Sie zum Auflösen von Konflikten quadratisches Sondieren mit $c = \frac{1}{2}$ und $d = \frac{1}{2}$. Notieren Sie die Ergebnisse der Streufunktion für die einzufügenden Elemente und ggf. die Ergebnisse der Sondierungsfunktion.

Lösung

1. Wir benutzen folgende Formel um Konflikte aufzulösen:

$$h'(x, i) = (h(x) + i)\%m$$

- Wir fügen nun 8 in die Hashtabelle ein. Dafür rechnen wir h'(8,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

$$h'(8, 0) = 0$$

  Da das Feld mit Index 0 noch frei ist, fügen wir 8 bei Index 0 in die Tabelle ein.
- Wir fügen nun 3 in die Hashtabelle ein. Dafür rechnen wir h'(3,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

$$h'(3, 0) = 3$$

  Da das Feld mit Index 3 noch frei ist, fügen wir 3 bei Index 3 in die Tabelle ein.
- Wir fügen nun 11 in die Hashtabelle ein. Dafür rechnen wir h'(11,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

$$h'(11, 0) = 3$$
$$h'(11, 1) = 4$$

  Da das Feld mit Index 4 noch frei ist, fügen wir 11 bei Index 4 in die Tabelle ein.
- Wir fügen nun 0 in die Hashtabelle ein. Dafür rechnen wir h'(0,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

$$h'(0, 0) = 0$$
$$h'(0, 1) = 1$$

  Da das Feld mit Index 1 noch frei ist, fügen wir 0 bei Index 1 in die Tabelle ein.
- Wir fügen nun 16 in die Hashtabelle ein. Dafür rechnen wir h'(16,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

$$h'(16, 0) = 0$$
$$h'(16, 1) = 1$$
$$h'(16, 2) = 2$$

  Da das Feld mit Index 2 noch frei ist, fügen wir 16 bei Index 2 in die Tabelle ein.

0	8
1	0
2	16
3	3
4	11
5	
6	
7	

Abbildung 1: Tabelle nach Einfügen und linearem Sondieren

2. Wir benutzen folgende Formel um Konflikte aufzulösen:

$$h'(x,i) = (h(x) + ci + di^2)\%m$$

- Wir fügen nun 8 in die Hashtabelle ein. Dafür rechnen wir h'(8,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

  $$h'(8,0) = 0$$

  Da das Feld mit Index 0 noch frei ist, fügen wir 8 bei Index 0 in die Tabelle ein.

- Wir fügen nun 3 in die Hashtabelle ein. Dafür rechnen wir h'(3,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

  $$h'(3,0) = 3$$

  Da das Feld mit Index 3 noch frei ist, fügen wir 3 bei Index 3 in die Tabelle ein.

- Wir fügen nun 11 in die Hashtabelle ein. Dafür rechnen wir h'(11,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

  $$h'(11,0) = 3$$

  $$h'(11,1) = 4$$

  Da das Feld mit Index 4 noch frei ist, fügen wir 11 bei Index 4 in die Tabelle ein.

- Wir fügen nun 0 in die Hashtabelle ein. Dafür rechnen wir h'(0,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

  $$h'(0,0) = 0$$

  $$h'(0,1) = 1$$

  Da das Feld mit Index 1 noch frei ist, fügen wir 0 bei Index 1 in die Tabelle ein.

- Wir fügen nun 16 in die Hashtabelle ein. Dafür rechnen wir h'(16,i) aus, bis wir ein freies Feld in der Hashtabelle finden.

  $$h'(16,0) = 0$$

  $$h'(16,1) = 1$$

  $$h'(16,2) = 3$$

  $$h'(16,3) = 6$$

  Da das Feld mit Index 6 noch frei ist, fügen wir 16 bei Index 6 in die Tabelle ein.

0	8
1	0
2	
3	3
4	11
5	
6	16
7	

Abbildung 2: Tabelle nach Einfügen und quadratischem Sondieren

Aufgabe 12: C0pbt

Prüfen Sie mittels der statischen Semantik von C0pbt (C0pb mit Typsystem), ob folgender Programmausschnitt unter der Typumgebung

$$\Gamma := \{x \mapsto \texttt{char},\ y \mapsto \texttt{char*},\ z \mapsto \texttt{int},\ p \mapsto \texttt{int**}\}$$

Typfehler aufweist.

```
{
    x = *y + z;
    *p = &z;
    x = **p;
}
```

Lösung

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{x} = \textbf{char}}{\Gamma \vdash \texttt{x} : \textbf{char}} \qquad \text{BinArith}\ \dfrac{\text{Indir}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{y} = \textbf{char*}}{\Gamma \vdash \texttt{y} : \textbf{char*}}}{\Gamma \vdash \texttt{*y} : \textbf{char}} \qquad \text{Var}\ \dfrac{\Gamma\ \texttt{z} = \textbf{int}}{\Gamma \vdash \texttt{z} : \textbf{int}}}{\Gamma \vdash \texttt{*y + z} : \textbf{int}} \qquad \dfrac{}{\text{char} \leftrightarrow \text{int}}}{\Gamma \vdash \texttt{x = *y + z;}}$$

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{p} = \textbf{int**}}{\Gamma \vdash \texttt{p} : \textbf{int**}} \qquad \text{Addr}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{z} = \textbf{int}}{\Gamma \vdash \texttt{z} : \textbf{int}}}{\Gamma \vdash \texttt{\&z} : \textbf{int*}} \qquad \dfrac{}{\text{int*} \leftrightarrow \text{int*}}}{\Gamma \vdash \texttt{*p = \&z;}}$$

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{x} = \textbf{char}}{\Gamma \vdash \texttt{x} : \textbf{char}} \qquad \text{Indir}\ \dfrac{\text{Indir}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{p} = \textbf{int**}}{\Gamma \vdash \texttt{p} : \textbf{int**}}}{\Gamma \vdash \texttt{*p} : \textbf{int*}}}{\Gamma \vdash \texttt{**p} : \textbf{int}} \qquad \dfrac{}{\text{char} \leftrightarrow \text{int}}}{\Gamma \vdash \texttt{x = **p;}}$$

$$\text{Seq}\ \frac{\Gamma \vdash \texttt{*p = \&z;} \qquad \Gamma \vdash \texttt{x = **p;}}{\Gamma \vdash \texttt{*p = \&z;}\sqcup\texttt{x = **p;}}$$

$$\text{Seq}\ \frac{\Gamma \vdash \texttt{x = *y + z;} \qquad \Gamma \vdash \texttt{*p = \&z; x = **p;}}{\Gamma \vdash \texttt{x = *y + z;}\sqcup\texttt{*p = \&z; x = **p;}}$$

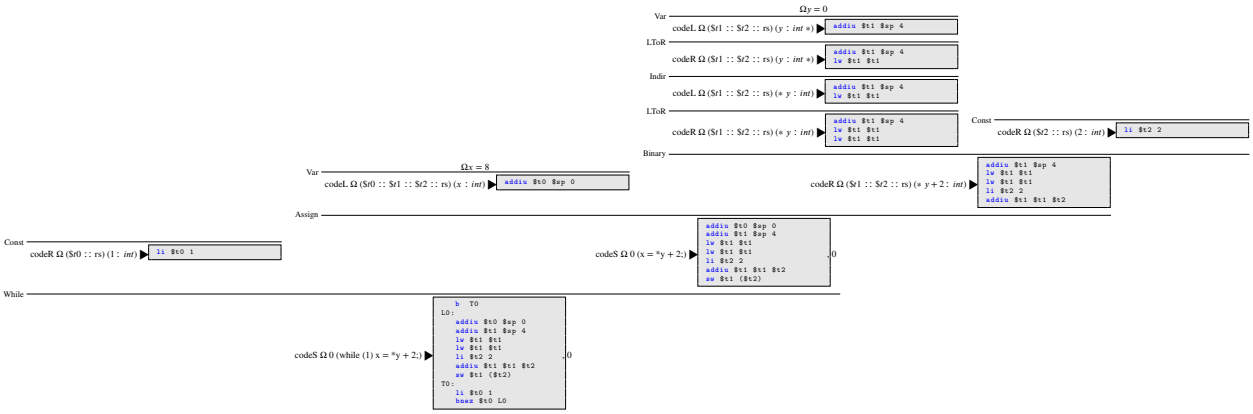
Das Programm weist unter der angegebenen Typumgebung Γ keine Typfehler auf.

Aufgabe 13: Codegenerierung

Generieren Sie MIPS Code für das folgende C0 Programm. Bauen Sie dazu den Inferenzbaum aus den Regeln zur Codegenerierung auf und schreiben Sie anschlieSSend die Codesequenz auf. Nehmen Sie $\Omega = \{x \mapsto 0, y \mapsto 4, z \mapsto 8\}$ als Umgebung an.

```
    while (1) x = *y + 2;
```

Lösung


Ωy = 0
Var
codeL Ω ($r1 :: $r2 :: rs) (y : int *)
addiu $t1 $sp 4
LToR
codeR Ω ($r1 :: $r2 :: rs) (y : int *)
addiu $t1 $sp 4
lw $t1 $t1
Indir
codeL Ω ($r1 :: $r2 :: rs) (* y : int)
addiu $t1 $sp 4
lw $t1 $t1
LToR
codeR Ω ($r1 :: $r2 :: rs) (* y : int)
addiu $t1 $sp 4
lw $t1 $t1
lw $t1 $t1
Const
codeR Ω ($r2 :: rs) (2 : int)
li $t2 2
Binary
codeR Ω ($r1 :: $r2 :: rs) (* y + 2 : int)
addiu $t1 $sp 4
lw $t1 $t1
lw $t1 $t1
li $t2 2
addiu $t1 $t1 $t2
Ωx = 8
Var
codeL Ω ($r0 :: $r1 :: $r2 :: rs) (x : int)
addiu $t0 $sp 0
Assign
Const
codeR Ω ($r0 :: rs) (1 : int)
li $t0 1
codeS Ω 0 (x = *y + 2;)
addiu $t0 $sp 0
addiu $t1 $sp 4
lw $t1 $t1
lw $t1 $t1
li $t2 2
addiu $t1 $t1 $t2
sw $t1 ($t2)
,0
While
codeS Ω 0 (while (1) x = *y + 2;)
b T0
L0:
addiu $t0 $sp 0
addiu $t1 $sp 4
lw $t1 $t1
lw $t1 $t1
li $t2 2
addiu $t1 $t1 $t2
sw $t1 ($t2)
T0:
li $t0 1
bnez $t0 L0
,0

Aufgabe 14: Korrektheitsbeweise

Überprüfen Sie mit Hilfe des *wp*-Kalküls, ob folgende Programme funktional korrekt sind.

(a) $[x \geq 0 \wedge y \geq 0]$

```
1 if (x - y >= 0)
2     x = x - y;
3 else
4     y = y - x;
```

$[x > y \vee y \geq x]$

(b) $[y \geq 0]$

```
1 x = y - 1;
2 r = x * (y + 1);
```

$[r + y \geq 0]$

(c) $[y > 4]$

```
1 x = y * 3;
2 if ((y + 2) < 6)
3     abort();
4 else
5     r = y + x;
```

$[r \geq 18]$

(d) $[y \geq 0]$

```
1 if (y > 4)
2     r = r * y;
3 else {
4     r = y + 1;
5     abort();
6 }
```

$[r \cdot y \neq 0]$

Lösung

a) wp(`if (x - y >= 0) x = x - y; else y = y - x; | x > y ∨ y >= x`)

$= (\texttt{x - y >= 0} \wedge wp(\texttt{x = x - y; | x > y} \vee \texttt{y >= x})) \vee (\texttt{x - y < 0} \wedge wp(\texttt{y = y - x; | x > y} \vee \texttt{y >= x}))$

$= (\texttt{x - y >= 0} \wedge (\texttt{x - y > y} \vee \texttt{y >= x - y})) \vee (\texttt{x - y < 0} \wedge (\texttt{x > y - x} \vee \texttt{y - x >= x}))$

$\Leftrightarrow$ `true`

b) wp(`x = y - 1; r = x * (y + 1); | r + y >= 0`)

$= wp(\texttt{x = y - 1; } \mid wp(\texttt{r = x * (y + 1); | r + y >= 0}))$

$= wp(\texttt{x = y - 1; | x * (y + 1) + y >= 0})$

$= \texttt{(y - 1) * (y + 1) + y >= 0}$

$\Leftrightarrow \texttt{yš - 1 + y >= 0}$

$\Leftrightarrow \texttt{y >=} \frac{1}{2} \texttt{ * (}\sqrt{5}\texttt{ - 1)} \vee \texttt{y <=} \frac{1}{2} \texttt{ * (-1 -} \sqrt{5}\texttt{)} \quad \lightning \quad \texttt{y >= 0}$

$\Leftrightarrow$ `false`

c) wp(`x = y * 3; if ((y + 2) < 6) abort(); else r = y + x; | r >= 18`)

$= wp(\texttt{x = y * 3; } \mid wp(\texttt{if ((y + 2) < 6) abort(); else r = y + x; | r >= 18}))$

$= wp(\texttt{x = y * 3; } \mid (\texttt{(y + 2) < 6} \wedge wp(\texttt{abort(); | r >= 18})) \vee (\texttt{(y + 2) >= 6} \wedge wp(\texttt{r = y + x; | r >= 18}))$

$= wp(\texttt{x = y * 3; } \mid (\texttt{(y + 2) < 6} \wedge \texttt{false}) \vee (\texttt{(y + 2) >= 6} \wedge \texttt{(y + x) >= 18}))$

$= \texttt{(y + 2) >= 6} \wedge \texttt{(y + (y * 3)) >= 18}$

$\Leftrightarrow \texttt{y >= 4} \wedge \texttt{4 * y >= 18}$

$\Leftrightarrow \texttt{y >= 4,5}$

Da unsere Sprache C0 nur Integer zulässt, also nur $y \in \mathbb{Z}$ zulässig sind, ist das Programm funktional korrekt.

d) wp(`if (y > 4) r = r * y; else {r = y + 1; abort();} | r * y ≠ 0`)

$= (\texttt{y > 4} \wedge wp(\texttt{r = r * y; | r * y} \neq \texttt{0})) \vee (\texttt{y <= 4} \wedge wp(\texttt{r = y + 1; abort(); | r * y} \neq \texttt{0}))$

$= (\texttt{y > 4} \wedge \texttt{r * yš} \neq \texttt{0})) \vee (\texttt{y <= 4} \wedge wp(\texttt{r = y + 1; } \mid wp(\texttt{abort(); | r * y} \neq \texttt{0})))$

$= (\texttt{y > 4} \wedge \texttt{r} \neq \texttt{0})) \vee (\texttt{y <= 4} \wedge wp(\texttt{r = y + 1; | false}))$

$= (\texttt{y > 4} \wedge \texttt{r} \neq \texttt{0})) \vee (\texttt{y <= 4} \wedge \texttt{false})$

$\Leftrightarrow \texttt{y > 4} \wedge \texttt{r} \neq \texttt{0}$

Das Programm ist nicht funktional korrekt, da `y >= 0` $\not\Rightarrow$ `y > 4` $\wedge$ `r` $\neq$ `0`

Aufgabe 15: Totale Korrektheit

Im Folgenden sollen Sie die totale Korrektheit von einem kleinen Programm `P` zeigen. Finden Sie dazu zu jeder While-Schleife eine ausreichend starke Invariante `I` und eine Terminierungsfunktion `t`. Beweisen Sie danach die Gültigkeit der Invariante `I` und der Terminierungsfunktion `t`. Nutzen Sie dann beides, um die totale Korrektheit von `P` zu zeigen.
(Hinweis: Vergessen Sie nicht zu zeigen, dass die Vorbedingung eine Teilmenge der Menge `Wp` darstellt, welche wir mit Hilfe des wp-Kalküls erhalten.)

$[\ y > 0 \wedge res = 0\ ]$

```
y' = y;
while (y > 0) {
    res = res + y;
    y = y - 1;
}
```

$[\ res = \frac{y'(y'+1)}{2}\ ]$

Lösung

Zunächst eine kurze Übersicht, wie wir die totale Korrektheit eines Programms `P` mit Vorbedingung `V`, Nachbedingung `N` und Schleife `S'` mit Schleifenrumpf `S` und Bedingung `b` beweisen:

1. Suche eine Schleifeninvariante `I` für `S'` und beweise, dass es sich dabei um eine gültige Schleifeninvariante handelt (Zeige, dass $B \cap I \subseteq Wp(s \mid I)$ hält).
2. Finde eine Terminierungsfunktion `t` für `S'` und beweise, dass es sich dabei um eine gültige Terminierungsfunktion für `S'` handelt (Zeige, dass $\forall k \in \mathbb{N}.[b \wedge I \wedge 0 \leq t \leq k+1]\ S\ [I \wedge 0 \leq t \leq k]$ hält).
3. Beweise, dass das Programm nach der Schleife, mit Vorbedingung $I \wedge \neg b$, in der Nachbedingung `N` terminiert (Zeige, dass $[I \wedge \neg b]P_{nach}\ [N]$ hält).
4. Beweise, dass das Programm vor der Schleife, mit Vorbedingung `V`, in der Nachbedingung $I \wedge t \geq 0$ terminiert (Zeige, dass $[V]P_{vor}\ [I \wedge t \geq 0]$ hält).

Definiere V als gegebene Vorbedingung und N als Nachbedingung:

$$V := [y > 0 \wedge res = 0]$$

$$N := [res = \frac{y'(y'+1)}{2}]$$

Definiere P als das gesamte Programm, S als den Schleifenrumpf, b als die Schleifenbedingung, P_{vor} als Programm vor der Schleife:

$$S := \texttt{res = res + y; y = y - 1;}$$
$$b := \texttt{y > 0;}$$
$$P_{vor} := y' = y;$$

Bemerke, dass das Programm nach der Schleife endet, somit "existiert kein Programm P_{nach} nach der Schleife". Definiere die gewählte Schleifeninvariante I und Terminierungsfunktion t:

$$I := res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0$$

$$t := y$$

Hinweis: *Im folgenden sind Umformungen zur leichteren Erkennbarkeit grün markiert.*

1. Zeige, dass I eine Schleifeninvariante ist.
   Es ist also zu zeigen: $b \wedge I \implies wp(S \mid I)$

$$
\begin{aligned}
& wp(\texttt{res = res + y; y = y - 1;} \mid res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0) \\
= {} & wp(\texttt{res = res + y;} \mid res = \frac{y'(y'+1)}{2} - \frac{(y-1)((y-1)+1)}{2} \wedge y - 1 \geq 0) \\
= {} & res{+}y = \frac{y'(y'+1)}{2} - \frac{(y-1)((y-1)+1)}{2} \wedge y - 1 \geq 0 \\
\Leftrightarrow {} & res = \frac{y'(y'+1)}{2} - \frac{(y-1)y}{2}{+}y \wedge y > 0 \\
\Leftrightarrow {} & res = \frac{y'(y'+1)}{2} - \frac{y^2 - y + 2y}{2} \wedge y > 0 \\
\Leftrightarrow {} & res = \frac{y'(y'+1)}{2} - \frac{y^2 + y}{2} \wedge y > 0 \\
\Leftrightarrow {} & res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y > 0 \\
\Leftarrow {} & \underbrace{res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0}_{I} \wedge \underbrace{y > 0}_{b}
\end{aligned}
$$

   Es folgt, dass I eine Schleifeninvariante ist.

2. Zeige, dass t eine Terminierungsfunktion ist.
   Es ist also zu zeigen: $b \wedge I \wedge 0 \leq t \leq k+1 \implies wp(S \mid I \wedge 0 \leq t \leq k)$

$$
\begin{aligned}
& wp(\texttt{res = res + y; y = y - 1;} \mid res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0 \wedge 0 \leq y \leq k) \\
= {} & wp(\texttt{res = res + y;} \mid res = \frac{y'(y'+1)}{2} - \frac{(y-1)((y-1)+1)}{2} \wedge y - 1 \geq 0 \wedge 0 \leq y - 1 \leq k) \\
= {} & res{+}y = \frac{y'(y'+1)}{2} - \frac{(y-1)((y-1)+1)}{2} \wedge y - 1 \geq 0 \wedge 0 \leq y - 1 \leq k \\
\Leftrightarrow {} & \underbrace{res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2}}_{\text{siehe 1.}} \wedge 0 < y \leq k+1 \\
\Leftarrow {} & \underbrace{res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0}_{I} \wedge \underbrace{y > 0}_{b} \wedge 0 \leq y \leq k+1
\end{aligned}
$$

   Es folgt, dass t eine Terminierungsfunktion ist.

3. Zeige, dass das Programm nach der Schleife die Nachbedingung impliziert.
   Es ist also zu zeigen: $I \wedge \neg b \implies wp(P_{nach} \mid N)$
   Da das Programm aber direkt nach der Schleife endet, können wir direkt zeigen: $I \wedge \neg b \implies N$

$$
\begin{aligned}
& res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0 \wedge \neg(y > 0) \\
\Leftrightarrow {} & res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0 \wedge y \leq 0 \\
\Leftrightarrow {} & res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y = 0 \\
\Leftrightarrow {} & res = \frac{y'(y'+1)}{2} - 0 \\
\Leftrightarrow {} & \underbrace{res = \frac{y'(y'+1)}{2}}_{N}
\end{aligned}
$$

Folglich gilt die Aussage.

4. Zeige, dass das Programm vor der Schleife, mit Vorbedingung V, in der Nachbedingung $N = I \wedge t \geq 0$ terminiert.
   Es ist also zu zeigen: $V \implies wp(P_{vor} \mid I \wedge t \geq 0)$

$$
\begin{aligned}
& wp(y' = y; \mid res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0 \wedge y \geq 0) \\
\Leftrightarrow\ & wp(y' = y; \mid res = \frac{y'(y'+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0) \\
=\ & res = \frac{y(y+1)}{2} - \frac{y(y+1)}{2} \wedge y \geq 0 \\
\Leftrightarrow\ & res = 0 \wedge y \geq 0 \\
\Leftarrow\ & \underbrace{res = 0 \wedge y > 0}_{V}
\end{aligned}
$$

   Folglich gilt die Aussage.

Da alle Punkte gelten, folgt, dass das Programm total korrekt ist, also gilt `[V]P[N]`.

Aufgabe 16: Invarianten erkennen

Betrachten Sie die folgende Schleife:

```
while (r >= y) {
  q = q + 1;
  r = r - y;
}
```

Welche der folgenden Ausdrücke sind Invarianten? Versuchen Sie, intuitiv zu erkennen, was eine Invariante ist. Begründen Sie Ihre Antworten. Intuitiv bedeutet, dass Sie wp nicht formell durchrechnen müssen.

(a) $true$

(b) $false$

(c) $r \geq 0 \wedge y \geq 0$

(d) $r \geq 0 \wedge y > 0$

(e) $r > 0 \wedge y \geq 0$

(f) $q > 0$

(g) $q \geq 0$

(h) $q \leq 0$

(i) $x = q * y + r$

(j) $x > q * y + y$

(k) $x \geq q * y + y$

Hinweis: Variablen, die im Programm nicht vorkommen, können einen beliebigen Wert haben. Eine Invariante ist nur dann eine Invariante, wenn sie eine Invariante für alle möglichen Werte für diese Variablen ist.

Lösung

Zur Erinnerung: Eine Invariante ist ein Ausdruck i, sodass $[e \wedge i]\ s\ [i]$ gilt, wobei e die Bedingung aus dem Schleifenkopf und s der Schleifenrumpf sind.

(a) Ist eine Invariante, da vieles (insbesondere unser Schleifenrumpf) als Nachbedingung *true* hat.

(b) Ist eine Invariante, da alles als Vorbedingung $false$ hat, denn aus Falschheit folgt alles.

(c) Ist eine Invariante. Vor der Ausführung gilt $r \geq y \geq 0$. y verändert sich wärend der Ausführung des Rumpfes nicht, aber r wird um y kleiner. Da jedoch $r \geq y$, ist danach $r \geq 0$, sodass die Invariante weiterhin gilt.

(d) Ist eine Invariante. Vor der Ausführung gilt $r \geq y > 0$. y verändert sich wärend der Ausführung des Rumpfes nicht, aber r wird um y kleiner. Da jedoch $r \geq y$, ist danach $r \geq 0$, sodass die Invariante weiterhin gilt.

(e) Ist keine Invariante. Wenn am Anfang z.B. $r = y = 5$, gilt danach $r = 0, y = 5$, was nicht mehr Teil der Invariante ist.

(f) Ist eine Invariante. q wird nicht kleiner, und bleibt somit > 0.

(g) Ist eine Invariante. q wird nicht kleiner, und bleibt somit ≥ 0.

(h) Ist keine Invariante. Wenn am Anfang z.B. $q = 0$, gilt danach $q = 1$, was jedoch nicht ≤ 0.

(i) Ist eine Invariante. Sei x beliebig, aber fest. Nach der Ausführung haben wir $x = (q + 1) * y + r - y$, was zur Invariante äquivalent ist.

(j) Ist keine Invariante. Wenn am Anfang z.B. $x = 1, q = 0, y = 1$, gilt nach der Ausführung $x = 1, q = 1, y = 1$, und die Invariante gilt nicht mehr.

(k) Ist keine Invariante. Wenn am Anfang z.B. $x = 0, q = 0, y = 1$, gilt nach der Ausführung $x = 0, q = 1, y = 1$, und die Invariante gilt nicht mehr.

Aufgabe 17: Terminierungsfunktionen erkennen

Betrachten Sie die folgende Schleife:

```
while (r >= y) {
  q = q + 1;
  r = r - y;
}
```

Entscheiden Sie *intuitiv* für jedes Paar aus Invariante i und Ausdruck t, ob t eine Terminierungsfunktion (zur Invariante i) ist. Begründen Sie Ihre Antworten.

(a) $i := y > 0, t := r$

(b) $i := y \geq 0, t := r$

(c) $i := y > 0 \wedge x = q * y + r, t := r$

(d) $i := y > 0 \wedge x = q * y + r, t := r - 42$

(e) $i := y > 0 \wedge x = q * y + r, t := r + 42$

(f) $i := y > 0 \wedge x = q * y + r, t := q$

(g) $i := y > 0 \wedge x = q * y + r, t := x - q$

(h) $i := y > 0 \wedge x = q * y + r, t := x - q * y$

(i) $i := true, t := r$

(j) $i := false, t := r$

Lösung

Zur Erinnerung: Ein Ausdruck t ist eine Terminierungsfunktion (zu einer Invariante i) genau dann, wenn $[e \wedge i \wedge 0 \leq t \leq k]\, s\, [i \wedge 0 \leq t < k]$. Wir wissen bereits, dass i eine Invariante ist, deshalb bleibt noch zu zeigen, dass die Nachbedingungen $0 \leq t$ und $t < k$ jeweils gelten. k ist eine im Programm sonst nicht vorkommende Variable (hier tatsächlich k) und weiterhin beliebig, aber fest.

(a) Ist eine Terminierungfunktion. r bleibt ≥ 0, da es ja vor der Ausführung des Rumpfes bereits $\geq y$ (und somit > 0) war (ergibt sich aus dem Schleifenkopf e), und um y kleiner wird. Gleichzeitig ist es nach der Ausführung $< k$, da es vor der Ausführung $\leq k$ war und nun um $y > 0$ kleiner geworden ist, also mindestens 1 kleiner.

(b) Ist keine Terminierungsfunktion. Wenn am Anfang z.B. $y = 0, r = 5, k = 5$, dann gilt nach der Ausführung, dass $y = 0, r = 5, k = 5$, aber $5 \not< 5$. Wir können nicht zeigen, dass k schrumpft, da eben $y = 0$ möglich ist.

(c) Ist eine Terminierungsfunktion, siehe (a).

(d) Ist keine Terminierungsfunktion. Wenn am Anfang z.B. $y = 5, r = 42, x = 47, q = 1, k = 0$, was ja die Vorbedingung erfüllt, muss danach jedoch $r - y - 42 > 0$ gelten, was jedoch nicht gilt. Intuitiv kann die Terminierungsfunktion unter 0 fallen.

(e) Ist eine Terminierungsfunktion. Da vor dem Durchlauf des Rumpfes $y > 0 \wedge r \geq y \wedge 0 \leq r + 42 \leq k$ gilt, ist r zu Beginn bereits $\geq y > 0$, und auch nach einem Durchlauf des Rumpfes ≥ 0. Dann ist auch $r + 42 \geq 0$. Gleichzeitig wird $r + 42$ genau wie r jeden Durchlauf um $y > 0$, also um mindestens 1 kleiner.

(f) Ist keine Terminierungsfunktion. Sei $q = 5, k = 5$. Nach dem Durchlauf ist $q = 6, k = 5$, dann ist aber nicht $q < k$. Intuitiv ist q keine Terminierungsfunktion, da es nicht stetig kleiner wird.

(g) Ist eine Terminierungsfunktion. Da q jeden Durchlauf grösSer wird, wird $x - q$ jeden Durchlauf kleiner. Es bleibt zu zeigen, dass $x - q$ nicht kleiner als 0 wird. Dies kann nur passieren, wenn vor dem Durchlauf bereits $x - q = 0$. Dann ist jedoch $x = q$ und die Vorbedingung lautet $y > 0 \wedge q = q * y + r \wedge r \geq y$. Dies ist ein Widerspruch, da $r, y > 0$ sind. Dies liegt daran, dass wenn $x = q$ die Bedingung $r \geq y$ bereits verletzt ist, die Schleife also bereits terminiert hat.

(h) Ist eine Terminierungsfunktion. Da $x = q * y + r$, ist $r = x - q * y$. Da r eine Terminierungsfunktion zur Invariante i ist (siehe (c)), muss auch $x - q * y$ eine sein. Formell folgt das daraus, dass die Vorbedingung und Nachbedingung jeweils äquivalent zueinander sind in Verbindung mit Satz 6.1.1.

(i) Ist keine Terminierungsfunktion. Wenn am Anfang z.B. $r = 0, y = -2, k = 0$, so ist nach dem Durchlauf $r = 2$, und somit nicht $r < k$. Intuitiv können wir nicht zeigen, dass r schrumpft, da wir keine Aussage über den Wert von y treffen können.

(j) Ist eine Terminierungsfunktion. Wir haben in unserem Hoare-Tripel die Vorbedingung $false$, und da aus Falschheit alles folgt, ist das Hoare-Tripel gültig.

Aufgabe 18: Invarianten finden

Betrachten Sie die folgenden Programme. Finden Sie für die Schleifen jeweils eine ausreichend starke Invariante und eine dazu passende Terminierungsfunktion. Beweisen Sie dann, dass die Programme mit den gegebenen Vor- und Nachbedingungen total korrekt sind.

(a) Summe
$[n >= 0]$

```
sum = 0;
i = 0;
while (i < n) {
  i = i + 1;
  sum = sum + i;
}
```

$[sum + sum = n * (n+1)]$

(b) Potenzen
$[e \geq 0]$

```
x = 1;
y = e;
while (y > 0) {
  x = x * b;
  y = y - 1;
}
```

$[x = b^e]$

(c) Division
$[a \geq 0 \land b > 0]$

```
q = 0;
r = a;
while(r >= b)) {
  q = q + 1;
  r = r - b;
}
```

$[q \cdot b + r = a \land r \geq 0 \land q \geq 0 \land r < b \land b > 0]$

Lösung

(a) Wir verwenden als Invariante $I = 2 \cdot sum = i \cdot (i+1) \land i \leq n$ und als Terminierungsfunktion $t = n - i$. Wir definieren $b := i < n$ und $s :=$ `i = i + 1; sum = sum + i;`.

- Wir zeigen nun, dass I eine Invariante ist, also $b \land I \implies wp(s \mid I)$:

$$
\begin{aligned}
& wp(\texttt{i = i + 1; sum = sum + i;} \mid 2 \cdot sum = i \cdot (i+1) \land i \leq n) \\
=\ & wp(\texttt{i = i + 1;} \mid wp(\texttt{sum = sum + i;} \mid 2 \cdot sum = i \cdot (i+1) \land i \leq n)) \\
=\ & wp(\texttt{i = i + 1;} \mid 2 \cdot (sum + i) = i \cdot (i+1) \land i \leq n) \\
=\ & 2 \cdot (sum + (i+1)) = (i+1) \cdot ((i+1)+1) \land (i+1) \leq n \\
\Leftrightarrow\ & 2 \cdot sum + 2i + 2 = (i+1) \cdot (i+2) \land (i+1) \leq n \\
\Leftrightarrow\ & 2 \cdot sum + 2i + 2 = i \cdot (i+1) + 2 \cdot (i+1) \land (i+1) \leq n \\
\Leftrightarrow\ & 2 \cdot sum = i \cdot (i+1) \land (i+1) \leq n \\
\Leftarrow\ & 2 \cdot sum = i \cdot (i+1) \land i \leq n \land i < n
\end{aligned}
$$

- Wir zeigen, dass t eine Terminierungsfunktion ist, also $b \land I \land 0 \leq t \leq k+1 \implies wp(s \mid I \land 0 \leq t \leq k$:

$$
\begin{aligned}
& wp(\texttt{i = i + 1; sum = sum + i;} \mid 2 \cdot sum = i \cdot (i+1) \land i \leq n \land 0 \leq n - i \leq k) \\
=\ & wp(\texttt{i = i + 1;} \mid wp(\texttt{sum = sum + i;} \mid 2 \cdot sum = i \cdot (i+1) \land i \leq n \land 0 \leq n - i \leq k)) \\
=\ & wp(\texttt{i = i + 1;} \mid 2 \cdot (sum + i) = i \cdot (i+1) \land i \leq n \land 0 \leq n - i \leq k) \\
=\ & 2 \cdot (sum + (i+1)) = (i+1) \cdot ((i+1)+1) \land (i+1) \leq n \land 0 \leq n - (i+1) \leq k \\
\Leftrightarrow\ & 2 \cdot sum + 2i + 2 = (i+1) \cdot (i+2) \land (i+1) \leq n \land 0 \leq n - (i+1) \leq k \\
\Leftrightarrow\ & 2 \cdot sum + 2i + 2 = i \cdot (i+1) + 2 \cdot (i+1) \land (i+1) \leq n \land 0 \leq n - i \leq k+1 \\
\Leftrightarrow\ & 2 \cdot sum = i \cdot (i+1) \land (i+1) \leq n \land 0 \leq n - i \leq k+1 \\
\Leftarrow\ & 2 \cdot sum = i \cdot (i+1) \land i \leq n \land i < n \land 0 \leq n - i \leq k+1
\end{aligned}
$$

- Wir zeigen, dass das Programm nach der Schleife mit der Vorbedingung $I \wedge \neg b$ in der gegebenen Nachbedingung N terminiert. Da das Programm nach der Schleife leer ist, zeigen wir, dass $I \wedge \neg b \implies N$:

$$\begin{aligned} & 2 \cdot sum = i \cdot (i+1) \wedge i \leq n \wedge i \geq n \\ \Leftrightarrow\ & sum + sum = i \cdot (i+1) \wedge i = n \\ \Leftrightarrow\ & sum + sum = n \cdot (n+1) \end{aligned}$$

- Wir zeigen, dass das Programm vor der Schleife mit der gegebenen Vorbedingung V in der Nachbedingung $I \wedge t \geq 0$ terminiert:

$$\begin{aligned} & wp(\texttt{sum = 0; i = 0;} \mid 2 \cdot sum = i \cdot (i+1) \wedge i \leq n \wedge 0 \leq n - i) \\ =\ & wp(\texttt{sum = 0;} \mid wp(\texttt{i = 0;} \mid 2 \cdot sum = i \cdot (i+1) \wedge i \leq n \wedge 0 \leq n - i)) \\ =\ & wp(\texttt{sum = 0;} \mid 2 \cdot sum = 0 \cdot (0+1) \wedge 0 \leq n \wedge 0 \leq n - 0) \\ =\ & 2 \cdot 0 = 0 \cdot (0+1) \wedge 0 \leq n \wedge 0 \leq n - 0 \\ \Leftrightarrow\ & 0 = 0 \wedge 0 \leq n \\ \Leftarrow\ & n \geq 0 \end{aligned}$$

Somit ist das Programm mit der gegebenen Vorbedingung und der gegebenen Nachbedingung total korrekt.

(b) Wir verwenden als Invariante $I = y \geq 0 \wedge x = b^{e-y}$ und als Terminierungsfunktion $t = y$. Wir definieren $b := y > 0$ und $s :=$ `x = x * b; y = y - 1;`.

- Wir zeigen nun, dass I eine Invariante ist, also $b \wedge I \implies wp(s \mid I)$:

$$\begin{aligned} & wp(\texttt{x = x * b; y = y - 1;} \mid y \geq 0 \wedge x = b^{e-y}) \\ =\ & wp(\texttt{x = x * b;} \mid wp(\texttt{y = y - 1;} \mid y \geq 0 \wedge x = b^{e-y})) \\ =\ & wp(\texttt{x = x * b;} \mid (y-1) \geq 0 \wedge x = b^{e-(y-1)}) \\ =\ & (y-1) \geq 0 \wedge (x \cdot b) = b^{e-(y-1)} \\ \Leftrightarrow\ & y > 0 \wedge (x \cdot b) = b^{e-y} \cdot b \\ \Leftarrow\ & y \geq 0 \wedge x = b^{e-y} \wedge y > 0 \end{aligned}$$

- Wir zeigen, dass t eine Terminierungsfunktion ist, also $b \wedge I \wedge 0 \leq t \leq k+1 \implies wp(s \mid I \wedge 0 \leq t \leq k$:

$$\begin{aligned} & wp(\texttt{x = x * b; y = y - 1;} \mid y \geq 0 \wedge x = b^{e-y} \wedge 0 \leq y \leq k) \\ =\ & wp(\texttt{x = x * b;} \mid wp(\texttt{y = y - 1;} \mid y \geq 0 \wedge x = b^{e-y} \wedge 0 \leq y \leq k)) \\ =\ & wp(\texttt{x = x * b;} \mid (y-1) \geq 0 \wedge x = b^{e-(y-1)} \wedge 0 \leq y - 1 \leq k) \\ =\ & (y-1) \geq 0 \wedge (x \cdot b) = b^{e-(y-1)} \wedge 0 \leq y - 1 \leq k \\ \Leftrightarrow\ & y > 0 \wedge (x \cdot b) = b^{e-y} \cdot b \wedge 0 \leq y \leq k+1 \\ \Leftarrow\ & y \geq 0 \wedge x = b^{e-y} \wedge y > 0 \wedge 0 \leq y \leq k+1 \end{aligned}$$

- Wir zeigen, dass das Programm nach der Schleife mit der Vorbedingung $I \wedge \neg b$ in der gegebenen Nachbedingung N terminiert. Da das Programm nach der Schleife leer ist, zeigen wir, dass $I \wedge \neg b \implies N$:

$$\begin{aligned} & y \geq 0 \wedge x = b^{e-y} \wedge y \leq 0 \\ \Leftrightarrow\ & y = 0 \wedge x = b^{e} \end{aligned}$$

- Wir zeigen, dass das Programm vor der Schleife mit der gegebenen Vorbedingung V in der Nachbe-

dingung $I \wedge t \geq 0$ terminiert:

$$\begin{aligned}
& wp(\texttt{x = 1; y = e;} \mid y \geq 0 \wedge x = b^{e-y} \wedge 0 \leq y) \\
= {} & wp(\texttt{x = 1;} \mid wp(\texttt{y = e;} \mid y \geq 0 \wedge x = b^{e-y} \wedge 0 \leq y)) \\
= {} & wp(\texttt{x = 1;} \mid e \geq 0 \wedge x = b^{e-e} \wedge 0 \leq e) \\
= {} & e \geq 0 \wedge 1 = b^{e-e} \wedge 0 \leq e \\
\Leftrightarrow {} & e \geq 0 \wedge 1 = 1 \\
\Leftarrow {} & e \geq 0
\end{aligned}$$

(c) Wir verwenden als Invariante $I = q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0$ und als Terminierungsfunktion $t = r + 1$. Wir definieren $b := r \geq b$ und $s := \texttt{q = q + 1; r = r - b;}$.

- Wir zeigen nun, dass I eine Invariante ist, also $b \wedge I \implies wp(s \mid I)$:

$$\begin{aligned}
& wp(\texttt{q = q + 1; r = r - b;} \mid q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0) \\
= {} & wp(\texttt{q = q + 1;} \mid wp(\texttt{r = r - b;} \mid q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0)) \\
= {} & wp(\texttt{q = q + 1;} \mid q \cdot b + (r - b) = a \wedge (r - b) \geq 0 \wedge q \geq 0 \wedge b > 0) \\
= {} & (q + 1) \cdot b + (r - b) = a \wedge (r - b) \geq 0 \wedge (q + 1) \geq 0 \wedge b > 0 \\
\Leftrightarrow {} & q \cdot b + b + r - b = a \wedge r \geq b \wedge (q + 1) \geq 0 \wedge b > 0 \\
\Leftarrow {} & q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge r \geq b
\end{aligned}$$

- Wir zeigen, dass t eine Terminierungsfunktion ist, also $b \wedge I \wedge 0 \leq t \leq k+1 \implies wp(s \mid I \wedge 0 \leq t \leq k$:

$$\begin{aligned}
& wp(\texttt{q = q + 1; r = r - b;} \mid q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge 0 \leq r + 1 \leq k) \\
= {} & wp(\texttt{q = q + 1;} \mid wp(\texttt{r = r - b;} \mid q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge 0 \leq r + 1 \leq k)) \\
= {} & wp(\texttt{q = q + 1;} \mid q \cdot b + (r - b) = a \wedge (r - b) \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge 0 \leq r - b + 1 \leq k) \\
= {} & (q + 1) \cdot b + (r - b) = a \wedge (r - b) \geq 0 \wedge (q + 1) \geq 0 \wedge b > 0 \wedge 0 \leq r - b + 1 \leq k \\
\Leftrightarrow {} & q \cdot b + b + r - b = a \wedge r \geq b \wedge (q + 1) \geq 0 \wedge b > 0 \wedge 0 \leq r - b + 1 \leq k \\
\Leftarrow {} & q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge r \geq b \wedge 0 \leq r + 1 \leq k + 1
\end{aligned}$$

- Wir zeigen, dass das Programm nach der Schleife mit der Vorbedingung $I \wedge \neg b$ in der gegebenen Nachbedingung N terminiert. Da das Programm nach der Schleife leer ist, zeigen wir, dass $I \wedge \neg b \implies N$:

$$\begin{aligned}
& q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge r < b \\
\Leftrightarrow {} & q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge r < b \wedge b > 0
\end{aligned}$$

- Wir zeigen, dass das Programm vor der Schleife mit der gegebenen Vorbedingung V in der Nachbedingung $I \wedge t \geq 0$ terminiert:

$$\begin{aligned}
& wp(\texttt{q = 0; r = a;} \mid q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge 0 \leq r + 1) \\
= {} & wp(\texttt{q = 0;} \mid wp(\texttt{r = a;} \mid q \cdot b + r = a \wedge r \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge 0 \leq r + 1)) \\
= {} & wp(\texttt{q = 0;} \mid q \cdot b + a = a \wedge a \geq 0 \wedge q \geq 0 \wedge b > 0 \wedge 0 \leq a + 1) \\
= {} & 0 \cdot b + a = a \wedge a \geq 0 \wedge 0 \geq 0 \wedge b > 0 \wedge 0 \leq a + 1 \\
\Leftarrow {} & a \geq 0 \wedge b > 0
\end{aligned}$$