Programming 2 (SS 2023)
Saarland University
Faculty MI
Compiler Design Lab

# Sample Solution

Prof. Dr. Sebastian Hack
Pascal Lauer, M. Sc.
Marcel Ullrich, B. Sc.

The exam preparation sheet has been created by the Prog2 tutors. The exercises were selected, as the tutors classified them as as potential exam questions or as good preparation. **This does not guarantee that this kind of question will be found in the exam!** The extent of this sheet considerably exeeds the exam's extent. As most questions are completely new, it is possible that mistakes crept in. Please ask in the forum, if you are uncertain a solution is correct. We will try to fix it as fast as possible.

## Exercise 1: Multiple Choice

For each of the following statements decide if they are true or false. If you think that a statement is false try to explain to yourself why it is false.

1. Mips: The assembly directive `.data` marks the beginning of the code segment.
2. The Mips calling convention specifies that the stack pointer and the s-registers need to be restored by the called function before jumping back.
3. To load a word its address needs to be divisible by 4.
4. `int foo(int x);` is a valid function declaration in C.
5. In C it holds `sizeof(char)==2`.
6. `a == b` if `a.equals(b)`
7. Dividing by zero is undefined behaviour in Java.
8. It is not possible to create an object of an abstract class in Java.
9. The load factor of a hash table can be computed by dividing the number of inserted elements by the size of the hash table.
10. To run through every position in a hash table using quadratic probing it is not important to use a surjective hash function.
11. Black Box tests try to achieve full branch, path and code coverage.
12. `true` is the weakest statement.
13. $[a0] = \langle a \rangle \cdot 2$

## Solution

1. False, .text marks the beginning of the code segment.
2. True.
3. True.
4. True.
5. False, `sizeof(char)==1`.
6. False.
7. False, an exception is thrown.
8. True.

9. True.
10. False, if the function isn't surjective it is not possible to go through every position.
11. False, Black box testing tests if a program satisfies a specification.
12. True.
13. False, counterexample: $[10] = -2 \neq 2 = \langle 1 \rangle \cdot 2$

## Exercise 2:

Write a function `scream` that gets the address to a (null-terminated) ASCII string in `$a0` and prints this string in upper-case. You may assume that the string only contains letters.
Screaming is not just about saying it in upper-case, you also have to adjust your facial expressions and so on. For that reason, you should use our own `print_char` function which handles all of this. You can just give it a character in `$a0` and it will print it for you. Notice that it does more than that, so it uses lots of registers.

**Example:** Calling the function with `Hello World` results in printing `HELLO WORLD`.

**Hint:** The ASCII code of "A" is 65 and the ASCII code of "a" is 97.

Here is a start for your code. Notice, that you should only write the *TODO* part.

```
1  .text
2  main:
3      ...
4      jal      scream
5      ...
6
7  scream:
8      # TODO
9      jr       $ra
10
11 print_char:
12     ...
```

## Solution

The following solution also provides the `main` and `print_char` function to check your solution. But remember that the two functions could also look differently and use more registers. So your solution is only correct, if you respected the calling-convention.

```
1  .data
2  test_text:
3      .asciiz "Hello World"
4
5  .text
6  main:
7      la $a0 test_text
8      jal  scream
9
10     # end program
11     li $v0 10
12     syscall
13
14 scream:
15     # load character
```

```
16      lbu   $t0 ($a0)
17
18      # check for end of string
19      beq   $t0 0 end
20
21      # check if char is already upper-case
22      bltu $t0 97   no_convert
23
24      # change char to upper-case
25      subiu   $t0 $t0 32
26
27  no_convert:
28      # save used registers
29      addiu   $sp $sp -8
30      sw $a0 ($sp)
31      sw $ra 4($sp)
32
33      # print char
34      move $a0 $t0
35      jal   print_char
36
37      # restore used registers
38      lw $a0 ($sp)
39      lw $ra 4($sp)
40      addiu   $sp $sp 8
41
42      # increase current char address
43      addiu   $a0 $a0 1
44
45      # jump to beginning of loop
46      j   scream
47  end:
48      # return
49      jr $ra
50
51  print_char:
52      # print char
53      li $v0 11
54      syscall
55      jr $ra
```

## Exercise 3: Edit distance

The Levenshtein distance is a measure for the difference between two sequences. It is defined as the minimum number of single-character edits (insertions, deletions, substitutions) required to transform one sequence into another:

$$lev(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ lev(tail(a), tail(b)) & \text{if } a[0] = b[0], \\ 1 + min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b) \end{cases} & \text{otherwise} \end{cases}$$

where *tail* is a substring excluding the first character, and $x[n]$ is the $n$th character beginning at 0.
The implementation below uses dynamic programming to cache previously computed edit distances between the prefixes of the first sequence, and the prefixes of the second sequence.
Sequences are represented as byte strings, and the cache matrix is stored in row-major format as an array of words.

1. Fill in the gaps such that `min` returns the minimum of its three arguments.

```
# Calculates the minimum of three numbers.
# @param $a0 The first number.
# @param $a1 The second number.
# @param $a2 The third number.
# @return $v0 The smallest of the three numbers.
min:
    move    $v0 $a0
    ble     ___ ___  min__second_is_greater
    move    ___ ___
    min__second_is_greater:
    ble     ___ ___ min__third_is_greater
    move    ___ ___
    min__third_is_greater:
    jr      $ra
```

2. Next, complete the following function that initializes the cache matrix. Since any sequence can be transformed into an empty one by deleting all characters, the first row and column are set to the number of characters of the substring at that position.

```
# Initializes a given edit distance matrix.
# The elements of the first row/column are
# set to their respective column/row index.
# @param $a0 The address of the matrix (stored in row-major order).
# @param $a1 The number of rows of the matrix.
# @param $a2 The number of columns of the matrix.
initialize_matrix:
    # initialize the first row to [0, $a2)
    ____    $t0 0
    ____    initialize_matrix__first_row_loop_condition
    initialize_matrix__first_row_loop_body:
        sw      $t0 ($a0)
        addiu   $a0 $a0 _
        addiu   $t0 $t0 _
    initialize_matrix__first_row_loop_condition:
        blt     $t0 ___ initialize_matrix__first_row_loop_body

    # initialize the first column to [0, $a1)
    li      $t0 _
    mulu    $t1 $a2 _
    b       initialize_matrix__first_column_loop_condition
    initialize_matrix__first_column_loop_body:
        sw      $t0 ($a0)
        _____   $a0 $a0 $t1
        _____   $t0 $t0 1
    initialize_matrix__first_column_loop_condition:
        blt     $t0 ___ initialize_matrix__first_column_loop_body

    jr      $ra
```

3. Finally, fill out the missing parts such that `distance` adheres to the calling convention.

```
# Calculates the edit distance between two strings.
# @param $a0 The address of the first string.
# @param $a1 The length of the first string.
```

```
# @param $a2 The address of the second string.
# @param $a3 The length of the second string.
# @param 0($sp) The address of the result matrix.
# @return $v0 The edit distance between the first and the second string.
distance:
    # build stack frame
    addiu   ___ ___ ___
    sw      ___ 0($sp)
    sw      ___ 4($sp)
    sw      ___ 8($sp)
    sw      ___ 12($sp)
    sw      ___ 16($sp)
    sw      ___ 20($sp)
    sw      ___ 24($sp)
    sw      ___ 28($sp)
    sw      ___ 32($sp)

    # save arguments
    move    $s0 $a0
    addiu   $s1 $a1 1
    move    $s2 $a2
    addiu   $s3 $a3 1
    lw      $s4 36($sp)

    # initialize matrix
    move    $a0 $s4
    move    $a1 $s1
    move    $a2 $s3
    jal     initialize_matrix

    # fill matrix
    li      $s5 1
    mulu    $t0 $s3 4
    addu    $s7 $s4 $t0
    b       distance__outer_fill_loop_condition
    distance__outer_fill_loop_body:
        li      $s6 1
        b       distance__inner_fill_loop_condition
        distance__inner_fill_loop_body:
            addu    $t0 $s0 $s5
            lbu     $t0 -1($t0)
            addu    $t1 $s2 $s6
            lbu     $t1 -1($t1)
            sne     $t0 $t0 $t1

            mulu    $t1 $s3 4
            subu    $t1 $s7 $t1

            lw      $a0 ($s7)
            addiu   $a0 $a0 1

            lw      $a1 4($t1)
            addiu   $a1 $a1 1

            lw      $a2 ($t1)
            addu    $a2 $a2 $t0

            jal     min
            sw      $v0 4($s7)

            addiu   $s6 $s6 1
```

```
65              addiu   $s7 $s7 4
66          distance__inner_fill_loop_condition:
67              blt     $s6 $s3 distance__inner_fill_loop_body
68              addiu   $s5 $s5 1
69              addiu   $s7 $s7 4
70      distance__outer_fill_loop_condition:
71          blt     $s5 $s1 distance__outer_fill_loop_body
72
73      lw      $v0 -4($s7)
74
75      # reset stack frame
76      lw      ___ 0($sp)
77      lw      ___ 4($sp)
78      lw      ___ 8($sp)
79      lw      ___ 12($sp)
80      lw      ___ 16($sp)
81      lw      ___ 20($sp)
82      lw      ___ 24($sp)
83      lw      ___ 28($sp)
84      lw      ___ 32($sp)
85      addiu   ___ ___ ___
86
87      jr      $ra
```

## Solution

The complete code looks as follows:

```
 1 # Calculates the minimum of three numbers.
 2 # @param $a0 The first number.
 3 # @param $a1 The second number.
 4 # @param $a2 The third number.
 5 # @return $v0 The smallest of the three numbers.
 6 min:
 7     move    $v0 $a0
 8     ble     $v0 $a1 min__second_is_greater
 9     move    $v0 $a1
10     min__second_is_greater:
11     ble     $v0 $a2 min__third_is_greater
12     move    $v0 $a2
13     min__third_is_greater:
14     jr      $ra
15
16 # Initializes a given edit distance matrix.
17 # The elements of the first row/column are
18 # set to their respective column/row index.
19 # @param $a0 The address of the matrix (stored in row-major order).
20 # @param $a1 The number of rows of the matrix.
21 # @param $a2 The number of columns of the matrix.
22 initialize_matrix:
23     # initialize the first row to [0, $a2)
24     li      $t0 0
25     b       initialize_matrix__first_row_loop_condition
26     initialize_matrix__first_row_loop_body:
27         sw      $t0 ($a0)
28         addiu   $a0 $a0 4
29         addiu   $t0 $t0 1
30     initialize_matrix__first_row_loop_condition:
31         blt     $t0 $a2 initialize_matrix__first_row_loop_body
32
```

```
33      # initialize first column to [0, $a1)
34      li      $t0 1
35      mulu    $t1 $a2 4
36      b       initialize_matrix__first_column_loop_condition
37      initialize_matrix__first_column_loop_body:
38          sw      $t0 ($a0)
39          addu    $a0 $a0 $t1
40          addiu   $t0 $t0 1
41      initialize_matrix__first_column_loop_condition:
42          blt     $t0 $a1 initialize_matrix__first_column_loop_body
43
44      jr      $ra
45
46  # Calculates the edit distance between two strings.
47  # @param $a0 The address of the first string.
48  # @param $a1 The length of the first string.
49  # @param $a2 The address of the second string.
50  # @param $a3 The length of the second string.
51  # @param 0($sp) The address of the result matrix.
52  # @return $v0 The edit distance between the first and the second string.
53  distance:
54      # build stack frame
55      addiu   $sp $sp -36
56      sw      $ra 0($sp)
57      sw      $s0 4($sp)
58      sw      $s1 8($sp)
59      sw      $s2 12($sp)
60      sw      $s3 16($sp)
61      sw      $s4 20($sp)
62      sw      $s5 24($sp)
63      sw      $s6 28($sp)
64      sw      $s7 32($sp)
65
66      # save arguments
67      move    $s0 $a0
68      addiu   $s1 $a1 1
69      move    $s2 $a2
70      addiu   $s3 $a3 1
71      lw      $s4 36($sp)
72
73      # initialize matrix
74      move    $a0 $s4
75      move    $a1 $s1
76      move    $a2 $s3
77      jal     initialize_matrix
78
79      # fill matrix
80      li      $s5 1
81      mulu    $t0 $s3 4
82      addu    $s7 $s4 $t0
83      b       distance__outer_fill_loop_condition
84      distance__outer_fill_loop_body:
85          li      $s6 1
86          b       distance__inner_fill_loop_condition
87          distance__inner_fill_loop_body:
88              addu    $t0 $s0 $s5
89              lbu     $t0 -1($t0)
90              addu    $t1 $s2 $s6
91              lbu     $t1 -1($t1)
92              sne     $t0 $t0 $t1
93
```

```
94              mulu    $t1 $s3 4
95              subu    $t1 $s7 $t1
96
97              lw      $a0 ($s7)
98              addiu   $a0 $a0 1
99
100             lw      $a1 4($t1)
101             addiu   $a1 $a1 1
102
103             lw      $a2 ($t1)
104             addu    $a2 $a2 $t0
105
106             jal     min
107             sw      $v0 4($s7)
108
109             addiu   $s6 $s6 1
110             addiu   $s7 $s7 4
111         distance__inner_fill_loop_condition:
112             blt     $s6 $s3 distance__inner_fill_loop_body
113             addiu   $s5 $s5 1
114             addiu   $s7 $s7 4
115     distance__outer_fill_loop_condition:
116         blt     $s5 $s1 distance__outer_fill_loop_body
117
118     lw      $v0 -4($s7)
119
120     # reset stack frame
121     lw      $ra 0($sp)
122     lw      $s0 4($sp)
123     lw      $s1 8($sp)
124     lw      $s2 12($sp)
125     lw      $s3 16($sp)
126     lw      $s4 20($sp)
127     lw      $s5 24($sp)
128     lw      $s6 28($sp)
129     lw      $s7 32($sp)
130     addiu   $sp $sp 36
131
132     jr      $ra
```

## Exercise 4: RLE

Implement a function called `rle` that performs a run-length encoding of a given array, which replaces consecutive occurrences of the same element in an array with the count of occurrences and the value of the element.

`rle` should take a pointer `arr` to the first element of an array of `int`s and the length of this array `len` as input. You can assume that the array only contains non-negative numbers. If a number $x$, occurs consecutively $n \geq 2$ times in the array, `rle` should replace the n entries with two entries: $-n$ and $x$.

**Example:**

| **input:** | 1 | 1 | 1 | 8 | 4 | 4 | 4 | 4 |
|---|---|---|---|---|---|---|---|---|
| **output:** | -3 | 1 | 8 | -4 | 4 | ? | ? | ? |

`rle` should modify the input array in-place and return the length of the modified array. As a result of the modification, the remaining entries should shift forward. In that example above, the new length is 5. It is not necessary

to free the memory of the freed entries, and it doesn't matter which numbers are present in the elements beyond the new end of the array (indicated by ? in the example).

**Hint:** *It's unnecessary (but allowed) to use the dynamic memory allocation. If you used it, you have to deallocate the dynamically allocated memory.*

## Solution

```c
#include <stdlib.h>

int rle(int* arr, int len) {
   int cur = arr[0];
   int count = 1;
   int out = 0;
   for (int i = 1; i <= len; i++) {
      if (i != len? arr[i] == cur : 0) {
         count++;
      } else {
         if (count == 1) {
            arr[out++] = cur;
         } else {
            arr[out] = -count;
            arr[out+1] = cur;
            out += 2;
         }
         count = 1;
         if (i != len) cur = arr[i];
      }
   }
   return out;
}
```

## Exercise 5: Pointerchase

Determine by hand what output the following program will provide. Use the computer only to verify that your result is correct.

```c
#include <stdio.h>

int main() {
    int i = 4;
    int *ap, *bp;
    int **app, **bpp;

    int r[4] = {1,2,3,4};

    ap = &r[0];
    bp = &r[3];

    app = &ap;
    bpp = &bp;

    *ap = i;
    *(*(bpp)) = 11;

    while (*ap < 6) {
        bp--;
        if(*ap % 2 == 0) {
```

```
22              *(*bpp) = *ap + 25;
23          } else {
24              *bp = 7;
25          }
26          (*ap)++;
27      }
28
29      printf("%d-%d-%d-%d\n", r[0], r[1], r[2], r[3]);
30 }
```

## Solution

Output of the program: "6-7-29-11".

| Step | Line | i | ap | bp | app | bpp | r |
|---|---|---|---|---|---|---|---|
| 1 | 4 | – | – | – | – | – | – |
| 2 | 5 | 4 | – | – | – | – | – |
| 3 | 6 | 4 | ? | ? | – | – | – |
| 4 | 8 | 4 | ? | ? | ? | ? | – |
| 5 | 10 | 4 | ? | ? | ? | ? | [1,2,3,4] |
| 6 | 11 | 4 | &r[0] | ? | ? | ? | [1,2,3,4] |
| 7 | 13 | 4 | &r[0] | &r[3] | ? | ? | [1,2,3,4] |
| 8 | 14 | 4 | &r[0] | &r[3] | &ap | ? | [1,2,3,4] |
| 9 | 16 | 4 | &r[0] | &r[3] | &ap | &bp | [1,2,3,4] |
| 10 | 17 | 4 | &r[0] | &r[3] | &ap | &bp | [4,2,3,4] |
| 11 | 19 | 4 | &r[0] | &r[3] | &ap | &bp | [4,2,3,11] |
| 12 | 20 | 4 | &r[0] | &r[3] | &ap | &bp | [4,2,3,11] |
| 13 | 21 | 4 | &r[0] | &r[2] | &ap | &bp | [4,2,3,11] |
| 14 | 22 | 4 | &r[0] | &r[2] | &ap | &bp | [4,2,3,11] |
| 15 | 26 | 4 | &r[0] | &r[2] | &ap | &bp | [4,2,29,11] |
| 16 | 19 | 4 | &r[0] | &r[2] | &ap | &bp | [5,2,29,11] |
| 17 | 20 | 4 | &r[0] | &r[2] | &ap | &bp | [5,2,29,11] |
| 18 | 21 | 4 | &r[0] | &r[1] | &ap | &bp | [5,2,29,11] |
| 19 | 24 | 4 | &r[0] | &r[1] | &ap | &bp | [5,7,29,11] |
| 20 | 26 | 4 | &r[0] | &r[1] | &ap | &bp | [5,7,29,11] |
| 21 | 19 | 4 | &r[0] | &r[1] | &ap | &bp | [6,7,29,11] |
| 22 | 29 | 4 | &r[0] | &r[1] | &ap | &bp | [6,7,29,11] |

## Exercise 6:

1. Draw the AST for the following C0 program.

```
1 y = &x;
2 {
3   if (x > 5)
4     x = 10;
5   else
6     x = 0;
7
8   while(x > 10)
9     {*y = *y - 1;}
10 }
```

2. Consider the following program in C0pb:

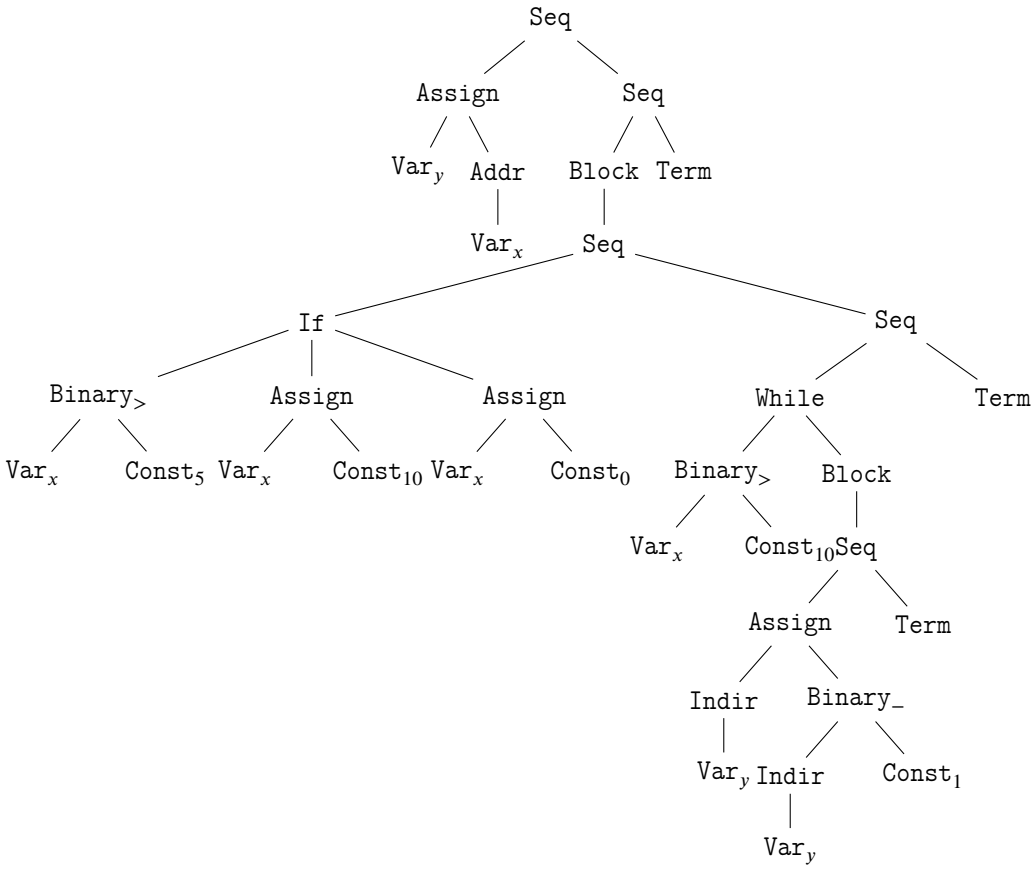
```
1 {
2     int x;
3     int* ptx;
4
5     x = 3;
6     ptx = &x;
7
8     while (x) {
9         int** y;
10        x = x - 1;
11        y = &ptx;
12        *ptx = -2 + **y;
13    }
14 }
```

(a) Type check the program using the type checking rules known from the lecture.

(b) Execute the program by using the execution rules for C0pb on the state $\sigma = (\{\}; \{\})$.

## Solution

1. The AST for the given program is:




2. Type-checking: The program is well-typed.

$$\dfrac{\dfrac{\Gamma' = \Gamma[\texttt{x} \mapsto \textbf{int}, \texttt{ptx} \mapsto \textbf{int*}] \quad \dfrac{\dfrac{\dfrac{}{\Gamma' \vdash \texttt{x = 3;}} \text{[TAssign]} \quad \text{Subtree 1}}{\Gamma' \vdash \texttt{x = 3; ptx = \&x; while (x) \{ } \textit{body} \texttt{ \}}} \text{[TSeq]}}{\Gamma \vdash \texttt{\{int x; int* ptx; x = 3; ptx = \&x; while (x) \{ } \textit{body} \texttt{ \}\}}} \text{[TBlock]} \quad \dfrac{}{\Gamma \vdash \varepsilon} \text{[TTerm]}}{\Gamma \vdash \texttt{\{int x; int* ptx; x = 3; ptx = \&x; while (x) \{ } \textit{body} \texttt{ \}\}}} \text{[TSeq]}$$

Subtree 1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma' \texttt{ x} = \textbf{int}}{\Gamma' \vdash \texttt{x} : \textbf{int}} \text{[TVar]}}{\Gamma' \vdash \texttt{\&x} : \textbf{int*}} \text{[TAddr]} \quad \dfrac{\Gamma' \texttt{ ptx} = \textbf{int*}}{\Gamma' \vdash \texttt{ptx} : \textbf{int*}} \text{[TVar]} \quad \dfrac{}{\texttt{int} * \leftrightarrow \texttt{int} *}}{\Gamma' \vdash \texttt{ptx = \&x;}} \text{[TAssign]} \quad \dfrac{\dfrac{\dfrac{\Gamma' \texttt{ x} = \textbf{int}}{\Gamma' \vdash \texttt{x} : \textbf{int}} \text{[TVar]}}{\Gamma' \vdash \texttt{while (x) \{ } \textit{body} \texttt{ \}}} \text{[TWhile]} \quad \text{Subtree 2}}{\Gamma' \vdash \texttt{ptx = \&x; while (x) \{ } \textit{body} \texttt{ \}}} \text{[TSeqS]}$$

Subtree 2:

$$\dfrac{\Gamma'' = \Gamma'[\texttt{y} \mapsto \textbf{int**}] \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma'' \texttt{ x} = \textbf{int}}{\Gamma'' \vdash \texttt{x} : \textbf{int}} \text{[TVar]} \quad \dfrac{-2^{31} \le 1 < 2^{31}}{\Gamma'' \vdash 1 : \textbf{int}} \text{[TConst]}}{\Gamma'' \vdash \texttt{x - 1} : \textbf{int}} \text{[TArith]} \quad \dfrac{\Gamma'' \texttt{ x} = \textbf{int}}{\Gamma'' \vdash \texttt{x} : \textbf{int}} \text{[TVar]} \quad \dfrac{}{\texttt{int} \leftrightarrow \texttt{int}}}{\Gamma'' \vdash \texttt{x = x - 1;}} \text{[TAssign]} \quad \text{Subtree3}}{\Gamma'' \vdash \texttt{x = x - 1; y = \&ptx; *ptx = -2 + **y;}} \text{[TSeq]}}{\Gamma' \vdash \texttt{\{int** y; x = x - 1; y = \&ptx; *ptx = -2 + **y;\}}} \text{[TBlock]}$$

Subtree 3:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma'' \texttt{ ptx} = \textbf{int*}}{\Gamma'' \vdash \texttt{ptx} : \textbf{int*}} \text{[TVar]}}{\Gamma'' \vdash \texttt{\&ptx} : \textbf{int**}} \text{[TAddr]} \quad \dfrac{\Gamma'' \texttt{ y} = \textbf{int**}}{\Gamma'' \vdash \texttt{y} : \textbf{int**}} \text{[TVar]} \quad \dfrac{}{\texttt{int **} \leftrightarrow \texttt{int **}}}{\Gamma'' \vdash \texttt{y = \&ptx;}} \text{[TAssign]} \quad \texttt{Subtree 4}}{\Gamma'' \vdash \texttt{y = \&ptx; *ptx = -2 + **y;}} \text{[TSeqS]}$$

Subtree 4:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma'' \texttt{ y} = \textbf{int**}}{\Gamma'' \vdash \texttt{y} : \textbf{int**}} \text{[TVar]}}{\Gamma'' \vdash \texttt{*y} : \textbf{int*}} \text{[TIndir]} \quad \dfrac{-2^{31} \le -2 < 2^{31}}{\Gamma \vdash \texttt{-2} : \textbf{int}} \text{[TConst]}}{\Gamma'' \vdash \texttt{-2 + **y} : \textbf{int}} \text{[TArith]} \quad \dfrac{\dfrac{\Gamma'' \texttt{ ptx} = \textbf{int*}}{\Gamma'' \vdash \texttt{ptx} : \textbf{int*}} \text{[TVar]}}{\Gamma'' \vdash \texttt{*ptx} : \textbf{int}} \text{[TIndir]} \quad \dfrac{}{\texttt{int} \leftrightarrow \texttt{int}}}{\Gamma'' \vdash \texttt{*ptx = -2 + **y;}} \text{[TAssign]}$$

3. Execution:
   Let s := `{int** y; x = x - 1; y = &ptx; *ptx = -2 + **y;}`,
   s1 := `x = x - 1; y = &ptx; *ptx = -2 + **y;`,
   s2 := `y = &ptx; *ptx = -2 + **y;`
   s3 := `*ptx = -2 + **y;`

$\langle$`{int x; int* ptx; x = 3; ptx = &x; while(x){s}}` $\mid \{\}; \{\}\rangle$
$\rightarrow$ $\langle$`x = 3; ptx = &x; while(x) s` ■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}; \{\triangle \mapsto ?, \pentagon \mapsto ?\}\rangle$ [Scope]
$\rightarrow$ $\langle$`ptx = &x; while(x) s` ■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}; \{\triangle \mapsto 3, \pentagon \mapsto ?\}\rangle$ [Assign]
$\rightarrow$ $\langle$`while(x) s` ■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}; \{\triangle \mapsto 3, \pentagon \mapsto \triangle\}\rangle$ [Assign]
$\rightarrow$ $\langle$ `s while(x)s` ■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}; \{\triangle \mapsto 3, \pentagon \mapsto \triangle\}\rangle$ [WhileTrue]
$\rightarrow$ $\langle$`s1` ■`while(x)s`■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}, \{\texttt{y} \mapsto \diamond\}; \{\triangle \mapsto 3, \pentagon \mapsto \triangle, \diamond \mapsto ?\}\rangle$ [Scope]
$\rightarrow$ $\langle$`s2` ■`while(x)s` ■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}, \{\texttt{y} \mapsto \diamond\}; \{\triangle \mapsto 2, \pentagon \mapsto \triangle, \diamond \mapsto ?\}\rangle$ [Assign]
$\rightarrow$ $\langle$`s3` ■`while(x)s` ■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}, \{\texttt{y} \mapsto \diamond\}; \{\triangle \mapsto 2, \pentagon \mapsto \triangle, \diamond \mapsto \pentagon\}\rangle$ [Assign]
$\rightarrow$ $\langle$■`while(x)s` ■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}, \{\texttt{y} \mapsto \diamond\}; \{\triangle \mapsto 0, \pentagon \mapsto \triangle, \diamond \mapsto \pentagon\}\rangle$ [Assign]
$\rightarrow$ $\langle$`while(x)s` ■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}; \{\triangle \mapsto 0, \pentagon \mapsto \triangle\}\rangle$ [Leave]
$\rightarrow$ $\langle$■ $\mid \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}; \{\triangle \mapsto 0, \pentagon \mapsto \triangle\}\rangle$ [WhileFalse]
$\rightarrow$ $\langle \epsilon \mid \{\}; \{\}\rangle$ [Leave]

The evaluation of *ptx = -2 + **y with
$\sigma = \langle \{\}, \{\texttt{x} \mapsto \triangle, \texttt{ptx} \mapsto \pentagon\}, \{\texttt{y} \mapsto \diamond\}; \{\triangle \mapsto 2, \pentagon \mapsto \triangle, \diamond \mapsto \pentagon\}\rangle$
is done in two steps:

1. The L-evaluation of *ptx:

$$\begin{aligned} L[\![\texttt{*ptx}]\!]\sigma &= R[\![\texttt{ptx}]\!]\sigma \\ &= \mu(L[\![\texttt{ptx}]\!]\sigma) \\ &= \mu(\rho(\texttt{ptx})) \\ &= \mu(\pentagon) \\ &= \triangle \end{aligned}$$

2. The R-evaluation of -2 + **y:

$$\begin{aligned} R[\![\texttt{-2 + **y}]\!]\sigma &= R[\![\texttt{-2}]\!] + R[\![\texttt{**y}]\!]\sigma \\ &= \texttt{-2} + R[\![\texttt{**y}]\!]\sigma \end{aligned}$$

Continue $R[\![\texttt{**y}]\!]\sigma$ for now:

$$\begin{aligned} R[\![\texttt{**y}]\!]\sigma &= \mu(L[\![\texttt{**y}]\!]\sigma) \\ &= \mu(R[\![\texttt{*y}]\!]\sigma) \\ &= \mu(\mu(L[\![\texttt{*y}]\!]\sigma)) \\ &= \mu(\mu(R[\![\texttt{y}]\!]\sigma)) \\ &= \mu(\mu(\mu(L[\![\texttt{y}]\!]\sigma))) \\ &= \mu(\mu(\mu(\rho\texttt{y}))) \\ &= \mu(\mu(\mu(\diamond))) \\ &= \mu(\mu(\pentagon)) \\ &= \mu(\triangle) \\ &= 2 \end{aligned}$$

Thus: $R[\![\texttt{-2}]\!] + R[\![\texttt{**y}]\!]\sigma = -2 + 2 = 0$

## Exercise 7:

Take a look at the following program:

```
1 int check(int a, int b, int c) {
2     int x = 2;
3
4     if (a > 0) {
5         x += 42;
6     }
7
8     if (b % 4 == 0) {
9         x += 4;
10    }
11
12    if (c != 0) {
13        int y = c * x;
```

```
14
15          if (y >= 0) {
16              return y;
17          } else {
18              return x;
19          }
20      } else {
21          return 0;
22      }
23 }
```

1. Write a test suite that has a *statement coverage* of 100%, i.e. all statements are covered by the test suite.
2. Extend your test suite, such that it has a *branch coverage* of 100%. If you think that no additional tests are required for this, explain in a sentence, why this is the case.
3. Extend your test suite again, such that it has a *path coverage* of 100%. If you think that no additional tests are required for this, explain in a sentence, why this is the case.

Now, look at the following program:

```
 1 int check2(int x, int y) {
 2     int v = x;
 3     if (v < 0) {
 4         v = -v;
 5     }
 6     if (v == 1337) {
 7         return 1;
 8     } else if (v > 1337) {
 9         if (y < 5) {
10             return 2;
11         }
12         return 3;
13     } else {
14         return 0;
15     }
16 }
```

4. Find out, what the *statement* coverage of each of the following calls to the function is. Write your answer as a fraction of the amount of covered statements and the amount of statements in the program.
   - `check2(-1337, 42)`
   - `check2(111, 6)`
   - `check2(2000, 6)`
   - `check2(1337, 2)`
   - `check2(-4321, 4)`
   - `check2(-1, 4000)`

## Solution

Test suites for subtasks 1, 2 and 3:

```
1 // Subtask 1: Statement Coverage
2
3 void test_both_c_positive() {
4     int res = check(1, 4, 10);
```

```
 5     assert(res == 480);
 6 }
 7
 8 void test_both_c_negative() {
 9     int res = check(1, 4, -10);
10     assert(res == 48);
11 }
12
13 void test_both_c_zero() {
14     int res = check(1, 4, 0);
15     assert(res == 0);
16 }
17
18 // Subtask 2: Branch Coverage
19
20 void test_none_c_positive() {
21     int res = check(0, 3, 10);
22     assert(res == 20);
23 }
24
25 // Subtask 3: Path Coverage
26
27 void test_none_c_negative() {
28     int res = check(0, 3, -10);
29     assert(res == 2);
30 }
31
32 void test_none_c_zero() {
33     int res = check(0, 3, 0);
34     assert(res == 0);
35 }
36
37 void test_first_c_positive() {
38     int res = check(1, 3, 10);
39     assert(res == 440);
40 }
41
42 void test_first_c_negative() {
43     int res = check(1, 3, -10);
44     assert(res == 44);
45 }
46
47 void test_first_c_zero() {
48     int res = check(1, 3, 0);
49     assert(res == 0);
50 }
51
52 void test_second_c_positive() {
53     int res = check(0, 4, 10);
54     assert(res == 60);
55 }
56
57 void test_second_c_negative() {
58     int res = check(0, 4, -10);
59     assert(res == 6);
60 }
61
62 void test_second_c_zero() {
63     int res = check(0, 4, 0);
64     assert(res == 0);
65 }
```

4. Statement Coverages:
    - `check2(-1337, 42)`: $\frac{5}{10}$ (Lines: `2,3,4,6,7`)
    - `check2(111, 6)`: $\frac{5}{10}$ (Lines: `2,3,6,8,14`)
    - `check2(2000, 6)`: $\frac{6}{10}$ (Lines: `2,3,6,8,9,12`)
    - `check2(1337, 2)`: $\frac{4}{10}$ (Lines: `2,3,6,7`)
    - `check2(-4321, 4)`: $\frac{7}{10}$ (Lines: `2,3,4,6,8,9,10`)
    - `check2(-1, 4000)`: $\frac{6}{10}$ (Lines: `2,3,4,6,8,14`)

## Exercise 8:

You have been hired by the university to implement part of the Mensa dishes in Java. Some dishes have already been implemented, so part of the infrastructure already exists (like the `Ingredients` but also the abstract class `MensaDish`).
Your task is to implement the following infrastructure, avoiding any kind of code duplication.
Create an interface called `Block`, which is implemented by the abstract class `MensaBlockDish`, which also inherits from the abstract class `MensaDish`.
Make sure that the following points are met:

1. A `Block` should always be able to return its volume (the volume in cubic centimetres should be represented here with an integer)
2. A `MensaBlockDish` has beside it's volume the same attriubutes as `MensaDish`
3. `NoodleCasserole`, `PotatoCasserole`, `PotatoGratin` and `Schales` are `MensaBlockDish`'es
4. The constructors of `NoodleCasserole`, `Schales`, `PotatoCasserole` and `PotatoGratin` should not take any paramenters
5. The respective volumes and prices are:
   `NoodleCasserole`: 480 $cm^3$, 2, 85€
   `PotatoCasserole`: 440 $cm^3$, 3, 15€
   `PotatoGratin`: 400 $cm^3$, 3, 35€
   `Schales`: 450 $cm^3$, 3, 45€
6. The dishes contain the following ingredients:
   `NoodleCasserole`: pasta, cream, cheese, eggs and onions
   `PotatoCasserole`: potatoes, onions, milk and eggs
   `PotatoGratin`: potatoes, cream, milk, butter, cheese and garlic
   `Schales`: potatoes, bacon and eggs
7. The `NoodleCasserole` tastes "bad", the `PotatoCasserole` tastes "good", the `PotatoGratin` is "very tasty" and the `Schales` are "edible"
8. You should be able to take a bite out of a `MensaBlockDish`, where 18 $cm^3$ are eaten.
   The method `bite` should print "finished" if the meal is empty after the bite and it should print "cry" if an attempt is made to take a bite from an empty meal.
9. Also, you were asked to add the functionality that you can print the ingredients from any `MensaDish`.
   The method should be named `printIngredients` and return the ingredients = {ingr_1, ..., ingr_n} as the string "ingr_1, ..., ingr_n"

```java
public enum Ingredient {
    Pasta("pasta"),
    Potatoes("potatoes"),
    Milk("milk"),
    Cheese("cheese"),
    Cream("cream"),
    Onions("onions"),
    Butter("butter"),
    Eggs("eggs"),
    Bacon("bacon"),
    Garlic("garlic");

    private String name;

    Ingredient(String name) {
        this.name = name;
    }

    public String toString() {
        return name;
    }
}
```

```java
public abstract class MensaDish {
    protected List<Ingredient> ingredients = new LinkedList<>();
    protected double price;
    protected String name;

    protected MensaDish(String name, double price,
                        List<Ingredient> ingredients) {
        this.name = name;
        this.price = price;
        this.ingredients.addAll(ingredients);
    }

    // should print how tasty the dish is
    public abstract void taste();

    public double getPrice() {
        return this.price;
    }

    public void eat() {
        System.out.println("eating " + name);
    }
}
```

## Solution

```java
public enum Ingredient {
    Pasta("pasta"),
    Potatoes("potatoes"),
    Milk("milk"),
```

```
    Cheese("cheese"),
    Cream("cream"),
    Onions("onions"),
    Butter("butter"),
    Eggs("eggs"),
    Bacon("bacon"),
    Garlic("garlic");

    private String name;

    Ingredient(String name) {
        this.name = name;
    }

    public String toString() {
        return name;
    }
}
```

```
import java.util.LinkedList;
import java.util.List;
import java.util.stream.Collectors;

public abstract class MensaDish {
    protected List<Ingredient> ingredients = new LinkedList<>();
    protected double price;
    protected String name;

    protected MensaDish(String name, double price,
                        List<Ingredient> ingredients) {
        this.name = name;
        this.price = price;
        this.ingredients.addAll(ingredients);
    }

    // should print how tasty the dish is
    public abstract void taste();

    public double getPrice() {
        return this.price;
    }

    public void eat() {
        System.out.println("eating " + name);
    }

    public void printIngredients() {
        List<String> ingredientNames =
                        ingredients.stream()
                                   .map(ingredient -> ingredient.toString())
                                   .collect(Collectors.toList());
        System.out.println(String.join(", ", ingredientNames));
    }
}
```

```
public interface Block {
    public int getVolume();
}
```

```
import java.util.List;

```

```java
public abstract class MensaBlockDish extends MensaDish implements Block {
    protected int volume;

    protected MensaBlockDish(String name, double price,
                             List<Ingredient> ingredients, int volume) {
        super(name, price, ingredients);
        this.volume = volume;
    }

    @Override
    public int getVolume() {
        return volume;
    }

    public void bite() {
        if (this.volume == 0) {
            System.out.println("cry");
            return;
        }
        if (this.volume <= 18) {
            this.volume = 0;
            System.out.println("finished");
            return;
        }
        this.volume -= 18;
    }
}
```

```java
public class NoodleCasserole extends MensaBlockDish {

    public NoodleCasserole() {
        super("noodle casserole", 2,85,
                List.of(Ingredient.Pasta, Ingredient.Cream, Ingredient.Cheese,
                        Ingredient.Eggs, Ingredient.Onions), 480);
    }

    @Override
    public void taste() {
        System.out.println("bad");
    }
}
```

```java
import java.util.List;

public class PotatoCasserole extends MensaBlockDish {
    public PotatoCasserole() {
        super("potato casserole", 3,15,
                List.of(Ingredient.Potatoes, Ingredient.Onions,
                        Ingredient.Milk, Ingredient.Eggs), 440);
    }

    @Override
    public void taste() {
        System.out.println("good");
    }
}
```

```java
import java.util.List;

public class PotatoGratin extends MensaBlockDish {
    public PotatoGratin() {
```

```
5         super("potato gratin", 3,35,
6                 List.of(Ingredient.Potatoes, Ingredient.Cream,
7                         Ingredient.Milk, Ingredient.Butter,
8                         Ingredient.Cheese, Ingredient.Garlic), 400);
9     }
10
11    @Override
12    public void taste() {
13        System.out.println("very tasty");
14    }
15 }
```

```
1 import java.util.List;
2
3 public class Schales extends MensaBlockDish {
4
5     public Schales() {
6     super("schales", 3,45,
7             List.of(Ingredient.Potatoes, Ingredient.Bacon,
8                     Ingredient.Eggs), 450);
9     }
10
11    @Override
12    public void taste() {
13        System.out.println("edible");
14    }
15 }
```

## Exercise 9: Inheritance Hierarchy

Which method is called in each case? State the output of the program.
**Note**: If you are unsure how to solve this exercise, you can find a detailed explanation in the forum.

## Class Hierarchy

```
1 public class A {
2   public void fun(B b) {
3     System.out.println("A.fun(B)");
4   }
5   public void fun(D d) {
6     System.out.println("A.fun(D)");
7   }
8 }
```

```
1 public class B extends A {
2   public void fun(A a) {
3     System.out.println("B.fun(A)");
4   }
5   public void fun(C c) {
6     System.out.println("B.fun(C)");
7   }
8   public void fun(E e) {
9     System.out.println("B.fun(E)");
10  }
11 }
```

```
1 public class C extends B {
2   public void fun(B b) {
3     System.out.println("C.fun(B)");
4   }
5 }
```

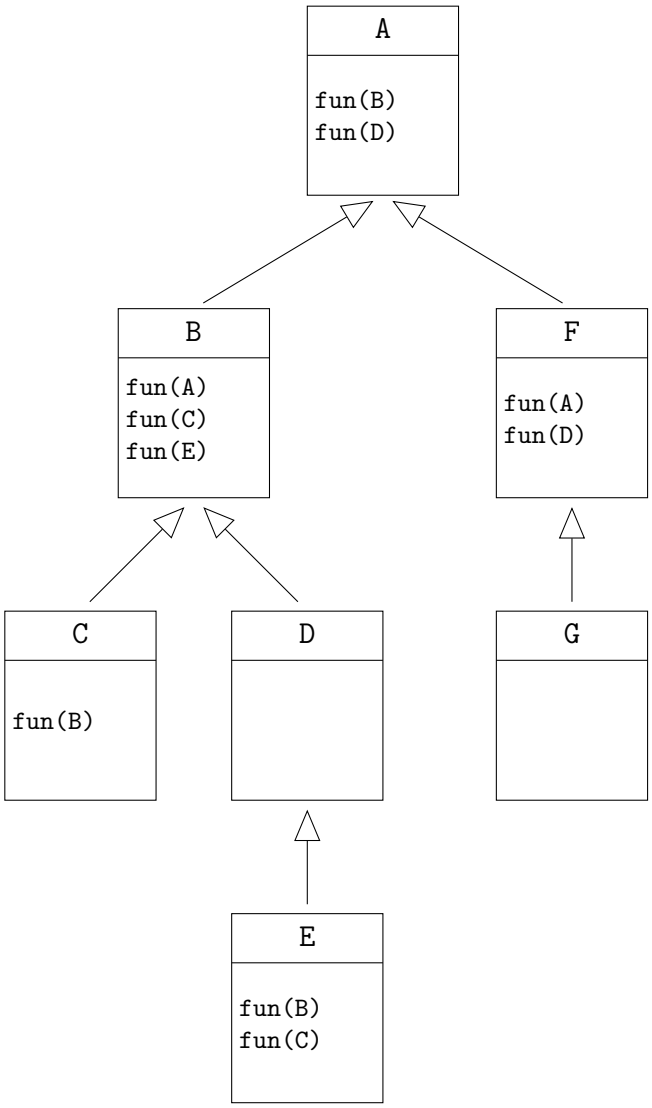
```
1 public class D extends B {
2 }
```

```
1 public class E extends D {
2   public void fun(B b) {
3     System.out.println("E.fun(B)");
4   }
5   public void fun(C c) {
6     System.out.println("E.fun(C)");
7   }
8 }
```

```
1 public class F extends A {
2   public void fun(A a) {
3     System.out.println("F.fun(A)");
4   }
5   public void fun(D d) {
6     System.out.println("F.fun(D)");
7   }
8 }
```

```
1 public class G extends F {
2 }
```

```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);
    }
}
```




## Solution

```
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); => Error!
```

```
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)
```

## Exercise 10:

Konrad Klug likes to read. The problem is that over the years he has accumulated so many books that he now has a hard time finding them in his bookshelves. To solve this problem, he has created the following class and implemented a hash function. Here, `digitSum(int x)` calculates the digit sum of `x`.

```
1 public class Book {
2
3     private String title;
4     private String author;
5     private int ISBN;
6
7     public int hashCode() {
8         return digitSum(ISBN);
9     }
10 }
```

The following is a table of some of Konrad's books that he wants to hash:

| Title | Author | ISBN |
|---|---|---|
| The Lord of the Rings | J. R. R. Tolkien | 978-0544273443 |
| The Art of Computer Programming | Donald E. Knuth | 978-0137935109 |
| The Hobbit | J. R. R. Tolkien | 978-0261103344 |
| The Ultimate Hitchhiker's Guide to the Galaxy | Douglas Adams | 978-0345453747 |
| Elements of Causal Inference | J. Peters, D. Janzing, B. Schölkopf | 978-0262037310 |
| The Chronicles of Narnia | C. S. Lewis | 978-0066238500 |
| Sherlock Holmes | Arthur Conan Doyle | 978-0141040288 |
| Artificial Intelligence: A Modern Approach | Stuart Russell, Peter Norvig | 978-0136042594 |
| Dracula | Bram Stoker | 978-0141439846 |

1. Hash the books in the above table into a `HashSet` of size 5. In case of collisions, use collision lists. Insert the

books in the same order as they appear in the table. What are the benefits and drawbacks of using collision lists?

2. Now hash the books into a `HashSet` of size 10 using linear probing. Insert the books in the same order as they appear in the table. What are the benefits and drawbacks of using linear probing? (*Hint:* The function for linear probing is $h(x, i) = (h'(x) + i) \mod m$)

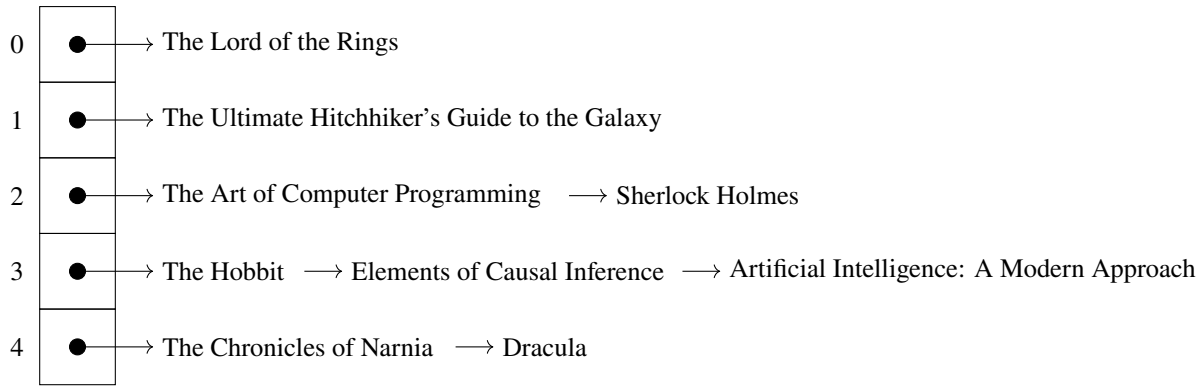
3. Now hash the books into a `HashSet` of size 10 using quadratic probing. Insert the books in the reverse order as they appear in the table. What are the benefits and drawbacks of using quadratic probing? (*Hint:* The function for quadratic probing is $h(x, i) = (h'(x) + \frac{i(i+1)}{2}) \mod m$)

## Solution

The hash codes of the books are calculated by adding up the digits of the ISBN.

| Title | Author | ISBN | Hashcode | mod 5 | mod 10 |
|---|---|---|---|---|---|
| The Lord of the Rings | ... | 978-0544273443 | 60 | 0 | 0 |
| The Art of Computer Programming | ... | 978-0137935109 | 62 | 2 | 2 |
| The Hobbit | ... | 978-0261103344 | 48 | 3 | 8 |
| The Ultimate Hitchhiker's Guide to the Galaxy | ... | 978-0345453747 | 66 | 1 | 6 |
| Elements of Causal Inference | ... | 978-0262037310 | 48 | 3 | 8 |
| The Chronicles of Narnia | ... | 978-0066238500 | 54 | 4 | 4 |
| Sherlock Holmes | ... | 978-0141040288 | 52 | 2 | 2 |
| Artificial Intelligence: A Modern Approach | ... | 978-0136042594 | 58 | 3 | 8 |
| Dracula | ... | 978-0141439846 | 64 | 4 | 4 |

1. Hashing the books using collision lists leads to the following `HashSet`:




The benefit of collision lists is that we can delete elements from the table easily. The drawback is that when the collision lists are long, lookup times increase. Further using collision lists can lead to a large memory overhead and bad cache locality.

2. Hashing the books using linear probing leads to the following `HashSet`:

| | |
|---|---|
| 0 | The Lord of the Rings |
| 1 | Artificial Intelligence: A Modern Approach |
| 2 | The Art of Computer Programming |
| 3 | Sherlock Holmes |
| 4 | The Chronicles of Narnia |
| 5 | Dracula |
| 6 | The Ultimate Hitchhiker's Guide to the Galaxy |
| 7 | |
| 8 | The Hobbit |
| 9 | Elements of Causal Inference |

One of the benefits of linear probing is that it has a good cache performance. The drawback of linear probing is that the hash table can get full, which means that we can't insert any more elements. Further, deleting elements from the table is not trivial. Finally, we can get the problem of clustering, in which case finding a spot for a new element can take a long time.

3. Hashing the books using quadratic probing leads to the following `HashSet`:

| | |
|---|---|
| 0 | The Lord of the Rings |
| 1 | The Hobbit |
| 2 | Sherlock Holmes |
| 3 | The Art of Computer Programming |
| 4 | Dracula |
| 5 | The Chronicles of Narnia |
| 6 | The Ultimate Hitchhiker's Guide to the Galaxy |
| 7 | |
| 8 | Artificial Intelligence: A Modern Approach |
| 9 | Elements of Causal Inference |

The befit of quadratic probing is that it fixes the problem of clustering that linear probing can lead to. The problem is that with quadratic probing, we are not guaranteed to find a spot for a new element, even if the table is not full. Further, deleting elements from the table is also not trivial.

## Exercise 11: Code Generation

Generate MIPS code for the following C0 programs. For that, draw an inference tree using the code generation derivation rules and give the MIPS code in the end. Use this offset function: $offs = \{x \mapsto 0, y \mapsto 4\}$.

1.
```
x = x + 2;
```

2.
```
while (y) x = x + 2;
```

You can reuse parts from previous exercises. If there are only small changes, mark them clearly in the derivation tree and the generated code.

## Solution

1.:

$$\text{[CSeq]}\ \frac{Helper \qquad \text{[CTerm]}\ \dfrac{}{codeP\ \ offs\ \ [\![\epsilon]\!] = c_{term}}}{codeP\ \ offs\ \ [\![x = \mathrm{x} + 2;]\!] = c_{assign}}$$

2.:

$$\text{[CSeq]}\ \frac{\text{[CWhile]}\ \dfrac{\text{[CLToR]}\ \dfrac{\text{[CVar]}\ \dfrac{offs\ \mathrm{y} = 4}{codeL\ \ offs\ \ \$\mathtt{t0} :: \$\mathtt{t1} :: \$\mathtt{t2} :: rs\ \ [\![\mathrm{y}]\!]\ \ \mathtt{int} = c_{var,2}} \quad L = \mathtt{lw}}{codeR\ \ offs\ \ \$\mathtt{t0} :: \$\mathtt{t1} :: \$\mathtt{t2} :: rs\ \ [\![\mathrm{y}]\!]\ \ \mathtt{int} = c_{ltor,2}} \qquad Helper}{codeS\ \ offs\ \ [\![\mathtt{while\,(y)}\, x = \mathrm{x} + 2;]\!] = c} \qquad \text{[CTerm]}\ \frac{}{codeP\ \ offs\ \ [\![\epsilon]\!] = c_{term}}}{codeP\ \ offs\ \ [\![\mathtt{while\,(y)}\, x = \mathrm{x} + 2;]\!] = c}$$

Helper:

$$\text{[CAssign]}\ \frac{\text{[CVar]}\ \dfrac{offs\ \mathrm{x} = 0}{codeL\ \ offs\ \ \$\mathtt{t0} :: \$\mathtt{t1} :: \$\mathtt{t2} :: rs\ \ [\![\mathrm{x}]\!]\ \ \mathtt{int} = c_{var,0}} \qquad \text{[CBinary]}\ \dfrac{\text{[CLToR]}\ \dfrac{\text{[CVar]}\ \dfrac{offs\ \mathrm{x} = 0}{codeL\ \ offs\ \ \$\mathtt{t1} :: \$\mathtt{t2} :: rs\ \ [\![\mathrm{x}]\!]\ \ \mathtt{int} = c_{var,1}} \quad L = \mathtt{lw}}{codeR\ \ offs\ \ \$\mathtt{t1} :: \$\mathtt{t2} :: rs\ \ [\![\mathrm{x}]\!]\ \ \mathtt{int} = c_{ltor,1}} \qquad \text{[CConst]}\ \dfrac{}{codeR\ \ offs\ \ \$\mathtt{t2} :: rs\ \ [\![2]\!]\ \ \mathtt{int} = c_{const,2}}}{codeR\ \ offs\ \ \$\mathtt{t1} :: \$\mathtt{t2} :: rs\ \ [\![\mathrm{x} + 2]\!]\ \ \mathtt{int} = c_{binary}} \quad S = \mathtt{sw}}{codeS\ \ offs\ \ [\![x = \mathrm{x} + 2;]\!] = c_{assign}}$$

$c_{const,2}$:

```
li $t2 2
```

$c_{var,0}$:

```
addiu $t0 $sp 0
```

$c_{var,1}$:

```
addiu $t1 $sp 0
```

$c_{var,2}$:

```
addiu $t0 $sp 4
```

$c_{ltor,2}$:

```
addiu $t0 $sp 4
lw $t0 $t0
```

$c_{ltor,1}$:

```
addiu $t1 $sp 0
lw $t1 $t1
```

$c_{binary}$:

```
addiu $t1 $sp 0
lw $t1 $t1
li $t2 2
addu $t1 $t1 $t2
```

$c_{assign}$:

```
addiu $t0 $sp 0
addiu $t1 $sp 0
lw $t1 $t1
li $t2 2
addu $t1 $t1 $t2
sw $t1 ($t0)
```

$c_{term}$:

```
# c_term
```

$c$:

```
  b   T
L:
    addiu $t0 $sp 0
    addiu $t1 $sp 0
    lw $t1 $t1
    li $t2 2
    addu $t1 $t1 $t2
    sw $t1 ($t0)
T:
    addiu $t0 $sp 4
    lw $t0 $t0
    bnez $t0 L

# c_term
```

## Exercise 12:

1. Carefully observe the following program and identify its functionality. Provide a specification, including the precondition and postcondition, followed by a formal proof.

```
1{ x = x + y;
2  y = x - y;
3  x = x - y; }
```

2. Take a look at the following program, find an invariant $I$ and verify that this program is correct with respect to the specification $(P, Q)$

```
1  while ((x + 1) * (x + 1) <=  y)
2  {
3    x = x + 1;
4  }
```

$$P := x = 0 \land y > 0 \quad Q := x * x \leq y < (x+1) * (x+1)$$

## Solution

1. The purpose of this program is to swap the values of two variables. The specifications are defined as follows:

$$P := X = x \land Y = y \quad Q := X = y \land Y = x$$

$$\text{[Seq]}\ \frac{\text{[Block]}\ \frac{\text{[Seq]}\ \frac{\text{[Consequence]}\ \frac{P \Rightarrow R_0 \qquad \text{[Assign]}\ \frac{}{\vdash \{R_0\}\ \texttt{x = x + y;}\ \{R_1\}}}{\vdash \{P\}\ \texttt{x = x + y;}\ \{R_1\}} \qquad \text{Tree}_1}{\vdash \{P\}\ \texttt{x = x + y; y = x - y; x = x - y;}\ \{Q\}}}{\vdash \{P\}\ \texttt{\{ x = x + y; y = x - y; x = x - y; \}}\ \{Q\}} \qquad \text{Tree}_2}{\vdash \{P\}\ \texttt{\{ x = x + y; y = x - y; x = x - y; \}}\ \{Q\}}$$

$\text{Tree}_1$:

$$\text{[SeqS]}\ \frac{\text{[Assign]}\ \frac{}{\vdash \{R_1\}\ \texttt{y = x - y;}\ \{R_2\}} \qquad \text{[Assign]}\ \frac{}{\vdash \{R_2\}\ \texttt{x = x - y;}\ \{Q\}}}{\vdash \{R_1\}\ \texttt{y = x - y; x = x = y;}\ \{Q\}}$$

$\text{Tree}_2$:

$$\text{[Term]}\ \frac{}{\vdash \{Q\}\ \epsilon\ \{Q\}}$$

and

$$R_2 = Q[x - y/x] = (X = y \land Y = x - y)$$
$$R_1 = R_2[x - y/y] = (X = x - y \land Y = x - (x - y)) = (X = x - y \land Y = y)$$
$$R_0 = R_1[x + y/x] = (X = (x + y) - y \land Y = y) = (X = x \land Y = y) = P \quad \text{so that, } P \Rightarrow R_0$$

2.

$$I = x * x \leq y \quad e = (x+1) * (x+1) \leq y$$

$$\text{[Consequence]}\ \frac{P \Rightarrow I \qquad \text{Tree}_1 \qquad I \land \neg e \Rightarrow Q}{\vdash \{P\}\ \texttt{while ((x + 1) * (x + 1) < y) \{x = x + 1;\}}\ \{Q\}}$$

By defining, $I \land \neg e = (x * x \leq y) \land (y < (x+1) * (x+1))$ it holds that $I \land \neg e$ implies $Q$. Proof of $P \Rightarrow I$ is also obviously.

Tree$_1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{I \wedge e \Rightarrow R_0 \qquad \dfrac{}{\vdash \{R_0\} \quad \texttt{x = x + 1;} \ \{I\}}\,[\text{Assign}]}{\vdash \{I \wedge e\} \quad \texttt{x = x + 1;} \ \{I\}}\,[\text{Consequence}]}{\vdash \{I \wedge e\} \quad \texttt{\{x = x + 1;\}} \ \{I\}}\,[\text{Block}]}{\vdash \{I\} \quad \texttt{while ((x + 1) * (x + 1) <= y) \{x = x + 1;\}} \ \{I \wedge \neg e\}}\,[\text{While}]$$

$R_0 = I[x+1/x] = (x+1) * (x+1) \leq y$

$I \wedge e = x * x \leq y \wedge (x+1) * (x+1) \leq y$ , so that $I \wedge e \Rightarrow R_0$