DM550: INTRODUCTION TO PROGRAMMING
Exercise list (Autumn 2021)

Part I: Fundamentals of Imperative Programming

1 Types, operators, variables and expressions

1. For each of the following expressions, write the order in which it is evaluated.

   (a) `a - b - c - d`

   (b) `a - b + c - d`

   (c) `a + b / c / d`

   (d) `a / b * c * d`

   (e) `a % b / c * d`

   (f) `(a - (b - c)) - d`

   (g) `a % (b % c) * d * e`

   (h) `(a + b) * c + d * e`

   (i) `(a + b) * (c - d) % e`

2. Consider the following variable declarations.

```
int a = 3;
double d = 2.19;
```

   Find the type and the value of each of the following expressions.

   (a) `a + 3 * a`

   (b) `(a + 3.0) * a`

   (c) `45 - a + 23`

   (d) `3.24 + a * 3`

   (e) `2 * 5.0 / a + 3`

   (f) `2 * 5 / a + 3`

   (g) `4 - d + a / 2`

   (h) `(d + 2) / a`

3. Suppose that `i` and `j` are two variables of a numeric type and that `b` is a variable of type `boolean`. Remove unnecessary parentheses from each of the following expressions.

   (a) `((3 * i) + 4) / 2`

   (b) `((3 * j) / (7 - i)) * (i + (-23 * j))`

   (c) `((((i + j) + 3) + j) * (((i - 4) / j) + -323))`

   (d) `(3 >= (j - 3)) == ((323 - (j * -7)) != (43))`

   (e) `((3 >= 5) == (!b || b))`

   (f) `(b || (!(b && (3 == (i * 2)))))`

   (g) `(!(!b) || (b && ((4 >= i+j) || (false))))`

4. For each of the following code snippets, find the value stored in each variable at the end of execution.

(a)

```
int j = 2, i = 1;
j = 3 + i * 2;
i = j / 2 * i + 3;
i = i + 1;
```

(b)

```
int i = 3;
double d = 3.0;
d = d - 2.3;
i = (int) d;
```

(c)

```
int x;
int y = 4;
x = y + y;
```

(d)

```
double b = 3.1, c = 0.0;
c = c + 2.0;
b = b * (c + 3.0);
int i = (int) (c + b);
i = i - 1;
```

(e)

```
int x = 5;
int y = x;
x = x + y;
```

(f)

```
int x;
int y = 4, z = 3;
x = y / z;
```

5. Suppose we need to work with the following data:

   - an age;
   - a weight;
   - the number of a lottery ticket;
   - a salary;
   - a person's gender (male or female);
   - a person's marital status (single, married, divorced, widowed);
   - a distance between stars, measured in light-years;
   - a distance on the Earth's surface, measured in meters.

   Propose names and types for variables to store these data.

2 Programming on numbers

1. Write a method `void printMultiples()` that prints on the screen the multiples of 7 that are less than 500.

2. Write a method `void printMultiples(int n)` that prints on the screen the multiples of 7 that are less than `n`.

3. Write a method `void printMultiples(int k, int n)` that prints on the screen the multiples of `k` that are less than `n`.

4. Write a method `int sumUpTo(int n)` to compute the sum of the natural numbers smaller than `n`.

5. Write a method `int sumBeyond(int k)` to find the least `n` such that the sum of the natural numbers smaller than `n` exceeds `k`.

6. Write a method `int sumBetween(int m, int n)` to compute the sum of the natural numbers larger than `m` and smaller than `n`.

7. Write a method `int sumEven(int n)` that computes the sum of all even numbers smaller than `n`.

8. Write a method `int factorial(int n)` that returns the factorial of `n`.

9. Write a method `int doubleFactorial(int n)` that returns `n`!! ($n!! = 1 \times 3 \times 5 \times \cdots \times n$, if n is odd, and $n!! = 2 \times 4 \times 6 \times \cdots \times n$, if n is even).

10. The sequence of Fibonacci numbers f_n is defined by $f(0) = f(1) = 1$ and $f(n+2) = f(n) + f(n+1)$. Write a method `int fibonacci(int n)` that returns the `n`-th Fibonacci number.

11. Write a method `int logarithm(int n)` that returns the integer base-2 logarithm of `n`.

12. Write a method `int countDivisors(int n)` that returns the number of divisors of `n`.

13. A perfect number is a number that equals the sum of its divisors (excluding itself). For example, 6 is a perfect number: its divisors are $\{1, 2, 3, 6\}$, and $1 + 2 + 3 = 6$.

    Write a method `boolean isPerfect(int n)` that checks whether `n` is a perfect number.

14. Write a method `int countPerfect(int n)` that returns the number of perfect numbers smaller than `n`.

15. Write a method `boolean isPrime(int n)` that checks whether `n` is prime.

16. Write a method `int countPrimes(int n)` that returns the number of primes smaller than `n`.

17. Write a method `int nthPrime(int n)` that returns the `n`-th prime.

18. Write a method `int largestDifference(int n)` that returns the largest difference between two consecutive primes smaller than `n`.

19. Write a method `int gcd(int m, int n)` that computes the greatest common divisor of `m` and `n` using Euclides' algorithm:

$$\begin{cases} \gcd(m,m) = m \\ \gcd(m,n) = \gcd(m,n-m) & m < n \\ \gcd(m,n) = \gcd(m-n,n) & m > n \end{cases}$$

20. Write a method `int lcm(int m, int n)` that returns the least common multiple of `m` and `n`.

21. Write a method `int firstDigit(int n)` that returns the first digit of the decimal representation of `n`.

22. Write a method `int firstDigitInBase(int n, int k)` that returns the first digit of the representation of `n` in base `k`.

23. Write a method `boolean isPalindrome(int n)` that checks whether `n` is a palindrome.

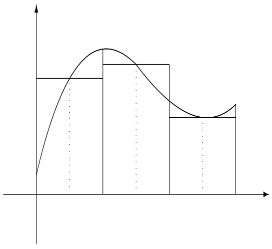
24. Write a method `int findPower(int k)` that returns the smallest number `n` such that $2^{\mathtt{n}}$ starts with `k`. What do you have to assume about `k`?

3 Small projects

1. *Solving equations.* Write a class `SolveEquation` to solve second-degree equations. The coefficients should be parameters of the `main` method. The program should print the solutions on the screen, if there are any, or a warning, otherwise.

   Recall that a second-degree equation has the general form $ax^2 + bx + c = 0$, where a, b and c are real numbers with $a \neq 0$. The solutions of this equation are $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$, assuming that $b^2 - 4ac > 0$. If $b^2 - 4ac = 0$, then there is only one solution ($x = -\frac{b}{2a}$), and if $b^2 - 4ac < 0$ then the equation has no (real-valued) solutions.

2. *Computing areas.* Suppose f is a continuous and positive function in an interval $[a,b]$. The area between the horizontal axis and the graph of f in the interval $[a,b]$ (also called the *integral* of f in $[a,b]$) can be computed as precisely as required by the following method: we divide the interval $[a,b]$ in n subintervals of equal width, and approximate the integral of f in each subinterval by the area of the rectangle whose height is given by the value of f value in the midpoint of the interval (see the figure below).

   Implement this method as a class `Integral` that prints on the screen the computed approximate value of the integral of f in the interval $[a,b]$. The function f is defined as a private method; for example, for $f(x) = x^2$, this class should include the following method.

   ```
   private static double f(double x){
     return x*x;
   }
   ```

   The values of a and b and the number of subintervals to use should be parameters of the `main` method.

4 Programming with arrays

1. Write a method `double sum(double[] v)` that computes the sum of all values in `v`.

2. Write a method `int zeros(int[] v)` that returns the number of zeros in `v`.

3. Write a method `int count(int[] v, int n)` that counts the number of occurrences of `n` in `v`.

4. Write a method `int smallerThan(int[] v, int n)` that returns the number of elements of `v` that are smaller than `n`.

5. Write a method `boolean member(int[] v, int n)` that checks whether `n` appears in `v`.

6. Write a method `boolean twoZeros(int[] v)` that checks whether `v` contains two consecutive zeros.

7. Write a method `String toString(int[] v)` that returns a textual representation of `v`.

8. Write a method `int[] squares(int n)` that returns an array with the squares of all natural numbers from 1 to `n`.

9. Write a method `int[] decreasingSquares(int n)` that returns an array with the squares of all natural numbers from `n` to 1.

10. Write a method `int[] divisors(int n)` that returns an array containing the divisors of `n`.

11. Write a method `double max(double[] v)` that returns the largest element in `v`.

12. Write a method `boolean subset(int[] v, int[] w)` that checks whether all elements of `v` occur in `w`.

13. Write a method `boolean setEquals(int[] v, int[] w)` that determines whether `v` and `w` represent the same set. Recall that a set does not have order and does not count duplicate elements.

14. Write a method `int[] intersection(int[] v, int[] w)` returning an array containing the elements that occur both in `v` occur in `w`.

15. Write a method `int firstPositionMax(int[] v)` that returns the index of the first occurrence of `v`'s maximum element.

16. Write a method `int lastPositionMax(int[] v)` that returns the index of the last occurrence of `v`'s maximum element.

17. Write a method `int addPositionsMax(int[] v)` that returns the sum of the indices of all the occurrences of `v`'s maximum element.

18. Write a method `int[] positionsMax(int[] v)` that returns an array containing the indices of the occurrences of `v`'s maximum element.

19. Write a method `void squareIt(double[] v)` that replaces each element in `v` by its square.

20. Write a method `void reverse(int[] v)` that reverses the values in `v` (i.e., it swaps the first element with the last, the second with the one before the last, . . .).

21. Write a method `int[] compare(int[] v, int n)` that returns an array containing: as first element, the number of elements of `v` larger than `n`; as second element, the number of elements of `v` equal to `n`; and, as third element, the number of elements of `v` smaller than `n`.

22. Write a method `int evenAfterSeven(int[] v)` that computes the number of even elements in `v` occurring after the first 7.

23. Write a method `int evenAfterLastSeven(int[] v)` that computes the number of even elements in `v` occurring after the last 7.

24. Write a method `int[] join(int[] v, int[] w)` that returns an array containing the elements of `v` followed by the elements of `w` (in the original order).

25. Write a method `int[] sortedJoin(int[] v, int[] w)` that takes two ordered arrays `v` and `w` as input and returns an ordered array containing all elements from either `v` or `w`.

26. Write a method `int[] shuffle(int[] v, int[] w)` that takes two arrays `v` and `w` and constructs an array by taking alternately one element from each of `v` and `w`.

27. Write a method `boolean isSorted(int[] v)` that checks whether the array `v` is sorted.

28. Write a method `int[] remove(int[] v, int n)` that returns an array containing all the elements of `v` that are different from `n`.

29. Write a method `int largestIncreasingSequence(int[] v)` that returns the length of the largest increasing sequence of consecutive elements of `v`.

30. The *sieve of Eratosthenes* is one of the oldest algorithms to find all prime numbers up to a given n. First, one writes down an array containing all numbers from 1 to n, and crosses out the 1. Next, one picks the next number k from the array that has not been crossed out, and crosses out all larger multiples of k. When the end of the array is reached, the numbers not crossed out are precisely the prime numbers smaller than or equal to n.

    Implement this algorithm as a method `int[] eratosthenes(int n)`. Use an efficient representation for the auxiliary array.

5 Programming with strings

It might be useful to recall that the lowercase alphabet uses ASCII codes 97 to 122, and the uppercase alphabet uses ASCII codes 65 to 90.

1. Write a method `int count(char c, String s)` that counts the number of occurrences of `c` in `s`.
2. Write a method `boolean member(char c, String s)` that checks whether `c` appears in `s`.
3. Write a method `boolean isPrefix(String s1, String s2)` that checks whether `s1` is a prefix of `s2`.
4. Write a method `boolean isSuffix(String s1, String s2)` that checks whether `s1` is a suffix of `s2`.
5. Write a method `boolean isSubstring(String s1, String s2)` that checks whether `s1` is a substring of `s2`.
6. Write a method `boolean contains(String s1, String s2)` that checks whether `s2` can be obtained from `s1` by deleting some characters.
7. Write a method `String toUppercase(String s)` that converts the string `s` to uppercase (ignoring all non-alphabetic characters).
8. Write a method `String toLowercase(String s)` that converts the string `s` to lowercase (ignoring all non-alphabetic characters).
9. Write a method `String toCamelCase(String s)` that converts a string of text into camel notation (i.e.: removes spaces and changes the first character after each space into uppercase, if it is a letter).
10. Write a method `boolean equals(String s1, String s2)` that determines whether two strings are equal.
11. Write a method `boolean equalsIgnoreCase(String s1, String s2)` that determines whether `s1` and `s2` are equal up to changes of case.
12. Write a method `int firstPosition(char c, String s)` that returns the index of the first occurrence of `c` in `s`, or -1 if `c` does not occur in `s`.
13. Write a method `int lastPosition(char c, String s)` that returns the index of the last occurrence of `c` in `s`, or -1 if `c` does not occur in `s`.
14. Write a method `int[] positions(char c, String s)` that returns an array containing the indices of the occurrences of `c` in `s`.
15. Write a method `boolean isPermutation(String s1, String s2)` that determines whether `s1` and `s2` contain exactly the same characters (counting repetitions).
16. Write a method `String reverse(String s)` that reverses a string.
17. Write a method `String reverseWords(String s)` that reverses the individual words inside a given string (preserving their order).

    Hint: write an auxiliary method `split` that splits a string at every occurrence of a particular character.
18. Write a method `String removeVowels(String s)` that takes a string as an argument and returns the result of removing all vowels in it.
19. Write a method `String respace(String s, int n)` that, given a string `s` and a positive integer `n`, returns the string obtained by first removing all spaces from `s` and afterwards adding a space after every `n` characters.

20. Write a method String shift(String s, int n) that receives a string and shifts it by the given number of characters.

21. Write a method String shiftWords(String s, int n) that shifts each individual word inside the argument string by the given number of characters.

22. Write a method String caesarCode(String s, int n) that increases the ASCII code of each character in s by n. What is the simplest way to implement the inverse method decode?

23. Write a method String encodeWithKey(String s, char[] code) that encodes the string s character-by-character. The 26-element table code indicates the codes for the uppercase letters A-Z, in order; lowercase characters should be encoded accordingly, and all remaining characters left unchanged.

24. Write a method char[] decode(char[] code) that generates the inverse code in the sense of the previous exercise. In other words, encodeWithKey(encodeWithKey(s,code), decode(code)) should return s.

25. Write a method int[] histogram(String s) that receives a string and returns an array of length 27 whose position i contains the number of occurrences of the i-th lettter of the alphabet (in either lower- or uppercase) in s. The first position (index 0) contains the total number of occurrences of non-alphabetic characters.

26. Write a method String replicate(String s, int[] v) that receives a string and an array of the same length and returns the string containing v[i] copies of the character s[i].

6 Programming with higher-dimension arrays

1. Write a method int[] dimensions(int[][] m) that returns an array with the lengths of all elements of m.

2. Write a method int[][] triangle(int n) that returns a triangular array of 1s where the first row has one element and each row afterwards contains one more element than the previous one.

3. Write a method int[][] multiplicationTable(int n) that returns a multiplication table up to n.

4. Reimplement methods sum, count, smallerThan, member, toString, max, squareIt and compare from section 3 so that their argument is now a bidimensional array (int[][] or double[][]).

5. Write a method void parity(int[][] m) that replaces each element in m by 0, if it is even, or 1, if it is odd.

6. Write a method int[][] differences(int[] v) that takes an array v and returns an array of arrays such that: its first line is v; and each other line contains the differences between consecutive elements of the previous lines. For example, for v={2,1,5,-2} the expected result of differences(v) is {{2,1,5,-2},{1,-4,7},{5,-11},{16}}.

7. Write a method int[][] pascal(int n) that returns the first n lines of Pascal's triangle: its first line is {1}, and every other line contains a 1, followed by the sums of all consecutive pairs of elements of the previous line, and a 1 at the end. For example, pascal(4) should return {{1},{1,1},{1,2,1},{1,3,3,1}}.

Matrices

8. A *matrix* is a two-dimensional array where all innermost arrays have the same size.

   Write a method boolean isMatrix(int[][] m) that checks whether m is a matrix.

9. Write a method int[] column(int[][] m, int j) that returns the j-th column of the matrix m. Which expression gives us the i-th row of m?

10. If two matrices have the same number of rows and columns, we can add them entry-by-entry. Write a method int[][] add(int[][] m1, int[][] m2) that implements this operation.

11. Write a method int[][] multiply(int a, int[][] m) that multiplies all entries of m by a.

12. Write a method boolean isSquareMatrix(int[][] m) that determines whether its argument is a square matrix.

13. Write a method `int trace(int[][] m)` that returns the sum of all elements in the diagonal of `m` (the *trace* of `m`), if `m` is a square matrix.

14. Write a method `int[][] zeros(int m, int n)` that returns a matrix with `m` rows and `n` columns whose entries are all 0.

15. Write a method `int[][] identity(int n)` that returns a matrix with `n` rows and `n` columns whose entries are 1 in the diagonal and 0 elsewhere.

    For example, `identity(3)` should return `{{1,0,0},{0,1,0},{0,0,1}}`.

16. Write a method `int[][] delRowAndCol(int[][] m, int i, int j)` that returns the matrix obtained by removing the `i`-th row and the `j`-th column of `m`.

17. Write a method `int[][] transpose(int[][] m)` that returns the matrix obtained from `m` by interchanging rows and columns.