Exam Computer Programming II 2021-10-25

Exam time: 14:00 – 19:00

Practical details and help during the exam:

- The teachers will often be available in https://uu-se.zoom.us/j/61112733698 during the exam You are welcome to spend the exam in that room, as during regular lessons.

  Do not ask questions in the "main room", but go to a breakout room and sign up in the document mentioned below.

- Keep the Google-document
  https://docs.google.com/spreadsheets/d/1dK1wM6d4YuikSL-IR5197I5z31XmI02xM_ATnlRh1EY/edit#gid=974818167
  open during the exam. We will post any late information about, for example, errors in the exam or answers to common questions there.

  Major changes will also be announced in Studium.

  You can also use the Google document to get in contact with the teachers. Fill in name and breakout-room number and we will show up.

- You can also put question on Slack, per email (sven-erik.ekstrom@it.uu.se or tom.smedsaas@it.uu.se), or by phone 0720575820 (Sven-Erik) or 0702544244 (Tom). Do not call but send a SMS. We will call you.

Submission:

- Submission of the exam takes place by uploading the Python files to the same place you downloaded the exam from (like regular assignments).

- The exam has A-tasks and B-tasks. A-tasks must be correctly answered to be regarded as passed. In case of code the code must solve the problem completely. B-tasks can give some credit even if they don't completely solve the problem, however we must be able to run the code!

- **Make a zip-file containing all of your files and upload it.** If you can't make zip-files you can also upload the files directly (preferably in one drag and drop operation). If you upload the sam file several times it is the last upload that is counted.
  Do not forget to press ''Upload"!

- If there are ant problkems with Studium you can e-mail your files to sven-erik.ekstrom@it.uu.se. Remember to include your exam registration code!

- **No submissions after 19:00 will be accepted!**

Rules

- You may not collaborate with anyone else during the exam. You may not copy code or text but you must write your answers yourself.
- You may use the Internet.
- You must write your solutions in the *in designated places* in the files `m1.py`, `m2.py`, `m3.py`, and `m4.py`.
  You must keep the names of the files, classes, methods and functions. Functions and methods must have the parameters as given in the task.
- You may not use other packages than those already imported in the files unless otherwise stated in the task. (The fact that a package is imported *does not* mean that it needs to be used!)
- Before your exam is approved, you may need to explain and justify your answers orally to a teacher.

PLEASE NOTE THAT WE ARE OBLIGATED TO NOTIFY ANY SUSPICION OF ILLEGAL COOPERATION OR COPYING AS A POSSIBLE ATTEMPT TO CHEAT!

Components of the exam:

The exam is divided into four sections, each with tasks corresponding to the four modules. There are A-tasks and B-tasks in all of the sections.

Grading:

3: At least eight A-tasks passed.

4: At least eight A-tasks passed and either two B-tasks passed or the voluntary assignment.

5: At least eight A-tasks passed and either all B-tasks or three B-tasks and the voluntary task.

Tasks in connection with module 1

The solution to this task must be written in the designated places in the file `m1.py`. The file also contains one `main`-function which tests the code. Feel free to add more tests!

A1: Write the function `length_longest(lst)` that returns the length of the longest sublist in `lst` (including `lst` itself). The function should handle sublists with *recursion*.

Example:
`length_longest(3)` $= 0$
`length_longest([])` $= 0$
`length_longest([2, 5]` $= 2$
`length_longest([1,2,[[1],[1,2]]])` $= 3$
`length_longest([1,2,[[1],[1,2,3,4]]])` $= 4$

A2: The function below rearranges the elements in the list that it gets as parameter:

```
def bubbelsort(aList):
    for i in range(len(aList)-1):
        for j in range(len(aList)-1):
            if aList[j] > aList[j+1]:
                aList[j], aList[j+1] = aList[j+1], aList[j]
```

(It is a well known but not particular efficient sorting algorithm. It not even the best implementation of it.)

How does the running time depend on the length n of the list? Answer with a Θ-expression.

You can either motivate your answer by theoretical analysis of the code or by systematic timing of the code. If you use timing, you have to include your code for that as well as the timing results.

B1: How does the time for the function below depend on the parameter `n`?

```
def foo(n):
    result = 1
    for k in range(3):
        for i in range(n*n):
            result += k*n
    return result
```

Answer with a Θ-expression.

You have to motivate your answer by timing the code.

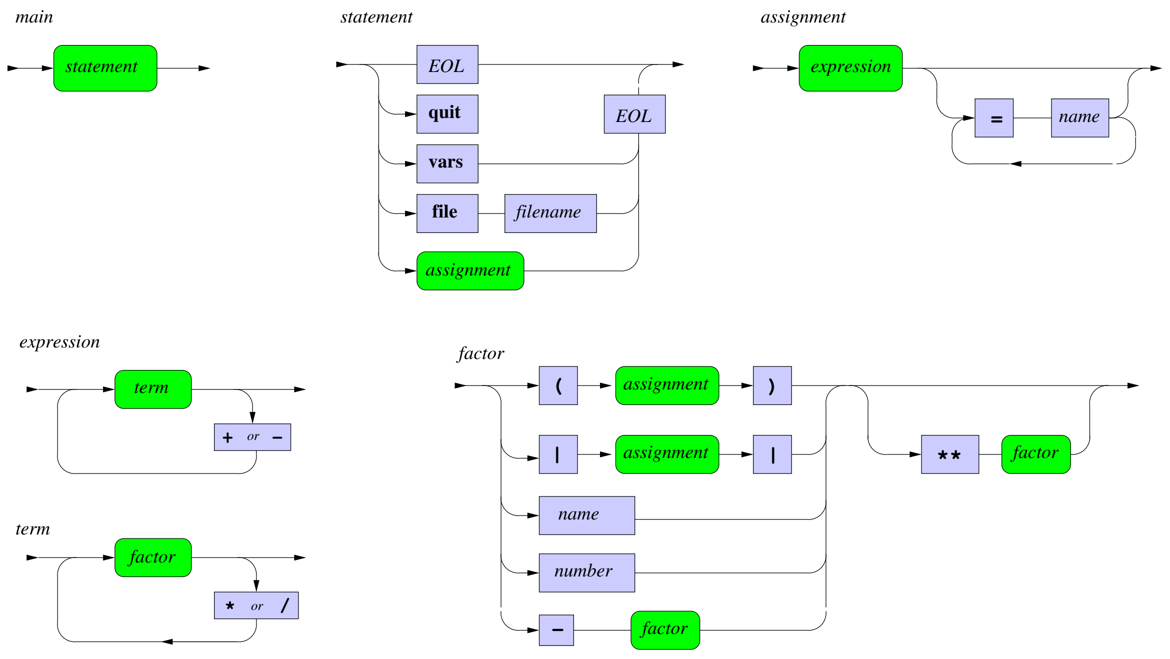
Estimate also how long time the call `foo(1000000)` would take.

Write your code for the timing in the `main`-function (or in a functions that is called from the `main` function) and include the output from your code.

Tasks in connection with module 2

The downloaded file m2.py implements a calculator like the one in the second assignment. Some constructions are removed and some added. Note that the downloaded file contains the class TokenizeWrapper.

The syntax is defined by the following charts:



The given code contains all functions (green panels) but all of them do not fully function.

The tasks are thus to modify the code so it is compliant with the charts and the description below. You may add more functions if you need.

The command `file` without parameter will cause the calculator to read input from the file `test.txt` which was included in the download.

The file begins with:

```
1 + -(2 - 3) = x     # 2.0
x/2 + x              # 3.0
(1=x) + (2=y)*(3=z)  # 7.0
```

The expected result stands after the comment character #.

After that comes test cases for the tasks which, of cause, don't work until you have solved the tasks.

A3: Make the program understand the vertical bars `||` for absolute value. Examples:

```
Input : file
File  : #### A3: Absolute values
File  :
File  : |3 - 2*3|           # 3.0
Result: 3.0
File  : -(3 - 2*3)          # 3.0
Result: 3.0
File  : -|3 - y*z|          # -3.0
Result: -3.0
File  : ||-3-4| - |-10=x||  # 3.0
Result: 3.0
File  : x                   # -10.0
Result: -10.0
File  : ||                  # Syntax error
*** Syntax error:  Expected number, word or '('
Error ocurred at '#␣Syntax␣error' just after '|'
File  : -|2-3-4 x           # Syntax error
*** Syntax error:  Expected '|'
Error ocurred at 'x' just after '4'
```

The syntax is defined by the charts.

A4: Introduce the operator `**` for ”raised to”. Example:

```
File  : 2**10               # 1024.0
Result: 1024.0
File  : 2**-1               # 0.5
Result: 0.5
File  : 2**3**2             # 512.0
Result: 512.0
File  : (2**3)**2           # 64.0
Result: 64.0
File  : (2=x)**(3*x=y)      # 64.0
Result: 64.0
File  : (-1)**(-2)          # 1.0
Result: 1.0
File  : 2****3              # Syntax error expected
*** Syntax error:  Expected number, word or '('
Error ocurred at '**' just after '**'
File  : 0**1                # 0.0
Result: 0.0
File  : 0**0                # Evaluation error
*** Evaluation error:  Illegal operation 0.0**0.0
File  : 0**-1               # Evaluation error
*** Evaluation error:  Illegal operation 0.0**-1.0
File  : (-2)**0             # 1.0
Result: 1.0
File  : 4**0.5              # 2.0
Result: 2.0
File  : (-4)**0.5           # Evaluation error
*** Evaluation error:  Illegal operation -4.0**0.5
```

The syntax chart shows where it should be implemented. (Note that the expression `x**y**z` is evaluated from right to left i.e. as `x**(y**z)` which actually is the most common interpretation programming languages)

The expression x^y (i.e. `x**y`) shall give *evaluation error* if $x = 0$ och $y \leq 0$ and if $x < 0$ and y doesn't have an integer value.

Hint: The tokenizer returns the sequence `122**3100` as three tokens: `'122'`, `'**'` and `'3100'`.

B2: In the given implementation side effects like assignments will be performed even if syntax or evaluation errors occur further ahead in the expression.

Examples:

```
Input : 1=x
Result: 1.0
Input : (9=x) +((2-3)* +*x
*** Syntax error:  Expected number, word or '('
Error ocurred at '+' just after '*'
Input : x
Result: 9.0
Input : (3=x)/(12-3*4)
*** Evaluation error:  Division by zero
Input : x
Result: 3.0
```

Modify the program so that no modifications will take place if the statement contains syntax or evaluation errors.

Thus, with the same input, the result should look like:

```
Input : 1=x
Result: 1.0
Input : (9=x) +((2-3)* +*x
*** Syntax error:  Expected number, word or '('
Error occurred at '+' just after '*'
Input : x
Result: 1.0
Input : (3=x)/(12-3*4)
*** Evaluation error:  Division by zero
Input : x
Result: 1.0
```

There are some evaluation errors that are difficult to foresee. Example:

```
Input : x
Result: 1.0
Input : (2=x) + 1/(2-x)
*** Evaluation error:  Division by zero
Input : x
Result: 2.0
```

You do not have to handle those.

In this task you may add parameters to the functions.

Tasks related to module 3

In this section you shall work with the file m3.py that contains the classes LinkedList, BST and ExamException.

There is also a main function with a few tests. You should add more!

The class LinkedList.py contains code for handling linked lists of objects. The lists are *not* sorted and the same value can occur several times. Integers are used in the examples but the code should work for all type of objects.

The class BST contains code for standard binary search trees.

The class ExamException is used for handling erroneous calls.

A5: In the given code there is a method for adding a new element at the end of the list:

```
    def append(self, x):
        self.first = self._append(self.first, x)
```

The method is using the help method _append. Write the help method!

The method should use *recursion* and should thus not contain any iteration.

A6: Objects from the class LinkedLists can be constructed from ordinary Python lists or any other iterable structure sent as an argument to the __init__-method:

```
 1      def __init__(self, param=None):
 2          self.first = None
 3          if param is None:
 4              return
 5          try:
 6              iter(param)
 7          except TypeError:
 8              self.first = self.Node(param)
 9              return
10
11          # Create the list from an iterable
12          self.first = self.Node(None)    # A dummy node
13          temp = self.first
14          for x in param:
15              temp.succ = self.Node(x)
16              temp = temp.succ
17          self.first = self.first.succ    # Remove the dummy node
```

An alternative to line 11-17 could be the following code:

```
        # Create the list from an iterable
        for x in param:
            self.append(x)
```

Both versions are in the downloaded code i.e. the method __init__ exists in two versions.

How does the running time depend of the length of the list in the respective cases?

Answer with a $\Theta(f(n))$-expression where n is the length of the list.

Write the answer with motivation in the designated place at the en of the file.

A7: The given code implements the method `__getitem__` which makes it possible to *get* the element from a certain index using the `[ ]`-operator.

Implement the method `__setitem__` which makes it possible to *change* a stored value at a specified index using the `[ ]`-operator!

Example:

```
    l = LinkedList([2, 5, 8])
    print(l)
    print(f'l[2]:␣{l[2]}')
    try:
        l[0] = 9
        l[2] = 11
        print(l)
        l[-1] = 17
    except ExamException as e:
        print(e)
```

Printouts:

```
(2, 5, 8)
l[2]: 8
(9, 5, 11)
Index out of range: -1
```

A8: Implement the method `is_omorph(self, other)` in the class `BST` so that it returns `True` if the own tree and the other tree have exactly the same shape, regardless of the key values, otherwise `False`.

Example:

```
    t1 = BST([1,2,3])
    t2 = BST([2,1,3])
    t3 = BST([2,3,1])
    t4 = BST([7,8,9])

    print(f't1␣and␣t2:␣{t1.is_omorph(t2)}')
    print(f't1␣and␣t3:␣{t1.is_omorph(t3)}')
    print(f't2␣and␣t3:␣{t2.is_omorph(t3)}')
    print(f't1␣and␣t4:␣{t1.is_omorph(t4)}')
```

Printouts:

```
t1 and t2: False
t1 and t3: False
t2 and t3: True
t1 and t4: True
```

B3: The method `merge` in the class `BST` inserts the keys from another search tree into this tree:

```python
    def merge(self, other_bst):
        for x in other_bst:
            self.insert(x)
```

In order to study the resulting tree the following code has been run:

```python
    print('     n   t1     t2     t3     t4')
    m = 1000
    for n in (1000, 2000, 4000, 8000, 16000,
              32000, 64000, 128000, 256000):
        t1 = random_tree(m)
        t2 = random_tree(n)
        h1 = t1.ipl()/m
        h2 = t2.ipl()/n
        t1.merge(t2)
        h3 = t1.ipl()/(n+m)
        t4 = random_tree(n+m)
        h4 = t4.ipl()/(n+m)
        print(f'{n:6d} {h1:5.1f}, {h2:5.1f}, {h3:5.1f}, {h4:5.1f}')
```

In the loop two trees t_1 and t_2 with m and n random keys respectively are created. Then the n keys from the t_2 are merged into t_1. The resulting tree, which is called t_3 in the printouts, thus contains $m + n$ keys.

Then, for comparison, a tree t_4 also with $m + n$ keys is created.

This is repeated for a sequence of different values of n and the average node height (i.e. the internal path length divided by the number of nodes) i listed.

This is the result from one run:

```
           Average node height
     n    t1     t2     t3     t4
  1000  11.3,  11.6,  12.7,  13.1
  2000  12.2,  13.3,  14.7,  14.0
  4000  11.9,  14.7,  16.7,  14.8
  8000  13.3,  15.4,  22.3,  15.8
 16000  13.8,  17.2,  29.7,  17.1
 32000  11.8,  20.8,  44.0,  18.7
 64000  11.5,  19.5,  74.5,  20.5
128000  11.5,  21.4, 140.7,  21.5
256000  10.8,  22.8, 271.8,  22.8
```

Explain why t_3 becomes so much worse than t_4 in spite of the fact that they have the same number of nodes and that they both are built from random numbers.

Rewrite `merge` som that the resulting tree t_3 does not become worse than t_4!

Remark: You may use other values of n as long as they give the same picture!

Tasks related to module 4

A9: A common type of matrix used in numerical computations has the form

$$M(\mathbf{x},n) = \begin{bmatrix} x_0^0 & x_0^1 & x_0^2 & \cdots & x_0^{n-1} \\ x_1^0 & x_1^1 & x_1^2 & \cdots & x_1^{n-1} \\ x_2^0 & x_2^1 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{m-1}^0 & x_{m-1}^1 & x_{m-1}^2 & \cdots & x_{m-1}^{n-1} \end{bmatrix}.$$

Here M is a function of $\mathbf{x}$, a vector (list) of length m such that $\mathbf{x} = [x_0, x_1, x_2, \ldots, x_{m-1}]$ and $n \geq 1$ is an integer.

If, for example, $\mathbf{x} = [x_0, x_1, x_2] = [5, 1.5, 3]$ we would have

$$M(\mathbf{x},3) = \begin{bmatrix} x_0^0 & x_0^1 & x_0^2 \\ x_1^0 & x_1^1 & x_1^2 \\ x_2^0 & x_2^1 & x_2^2 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 25 \\ 1 & 1.5 & 2.25 \\ 1 & 3 & 9 \end{bmatrix},$$

and

$$M(\mathbf{x},4) = \begin{bmatrix} x_0^0 & x_0^1 & x_0^2 & x_0^3 \\ x_1^0 & x_1^1 & x_1^2 & x_1^3 \\ x_2^0 & x_2^1 & x_2^2 & x_2^3 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 25 & 125 \\ 1 & 1.5 & 2.25 & 3.375 \\ 1 & 3 & 9 & 27 \end{bmatrix}.$$

Modify the function `matrix(x,n)` in `m4.py` such that it returns a list where each element in the list is itself a list with the contents of each row of the matrix $M(\mathbf{x}, n)$. **This should be done with a one line list comprehension.**

For the two examples above, you should return the following lists:

```
>>> x=[5, 1.5,  3]
>>> print(matrix(x,3))
[[1, 5, 25], [1.0, 1.5, 2.25], [1, 3, 9]]
>>> print(matrix(x,4))
[[1, 5, 25, 125], [1.0, 1.5, 2.25, 3.375], [1, 3, 9, 27]]
```

You are not allowed to import any extra packages or modules to solve this problem.

A10: The function `dice(n)` in the file `m4.py` simulates a "broken" dice, that insead of having the sides $\{1, 2, 3, 4, 5, 6\}$, has the sides $\{1, 2, 3, 4, 5, 5\}$. The argument `n` is how many many times the dice is thrown, and the function returns a list with `n` random values that the dice gave for every throw. **You should not modify this function.**

Modify the function `dice_average()` in the file `m4.py` such that it does the following:

- Run the function `dice(n)` with $n = 100000$, twenty times in parallel, that is, in total 2000000 "throws". You can use any parallel package or module you want, the important thing is not to speed up the execution.
- Return (i.e., do not print) the mean value for all the throws (i.e., only one value).

You are not allowed to import any extra packages or modules to solve this problem.

B4: You should write a program that creates thumbnails (downscaled size) of all image files that are in the same directory as the Python program you are running. You can use any package or module you want in this task. Make sure you read the tips given in the end.

Modify the function `thumbnail()` in `m4.py`, and you can write additional functions if you want to.

The program should

- work both on `.png` and `.jpg` files automatically;
- not have any in parameters (i.e., it should find all image files by itself);
- generate thumbnails with the same ratio between number of pixels of width and hight as the original image, but with the maximum of the two being 100 pixels;
- save the thumbnails in a directory with the name `thumb`. f the directory does not exist when the program is run, then the directory should be created. Add `thumb_` to the original file name, e.g., `fish.png` becomes `thumb_fish.png`;
- not have any hard coded "paths" where files are located on your own hard drive;
- executed in parallel (i.e., the creation of thumbnails is done in parallel).

Tips:

- Among the downloaded files for this exam are three example images that you can try you program on (`giraffe.jpg`, `moose.jpg`, and `fish.png`). The program should work on arbitrary filenames and number of files.
- The package `pillow` can manipulate images, and we recommend that you use it. After installation of `pillow`, you import it with: `from PIL import Image` (You can use other packages than this one if you want to).
- In the package `os` you have, for example, functions to find names of files in a directory.
- Start with getting the code to work without parallelization.