Tentamen på Programmeringsteknik II 2021-10-25

Skrivtid: 14:00 – 19:00

Observera: Denna skrivning är avsedd för de studenter som läst kursen ht 2020 eller senare.

Praktiska detaljer och hjälp under tentamen:

- Under stor del av tentan kommer lärare att finnas tillgängliga i Zoom: https://uu-se.zoom.us/j/61112733698. Du får gärna vara i ett breakout rum under tentan som under en vanlig lektion, men det är inte ett krav.

  Ställ inga frågor i "huvudrummet", utan gå till ett breakout room och skriv upp dig i dokumentet som beskrivs i punkten nedan.

- Ha Google-dokumentet
  https://docs.google.com/spreadsheets/d/1dK1wM6d4YuikSL-IR5197I5z31XmIO2xM_ATnlRh1EY/edit#gid=974818167
  öppet under tentans gång. Där kommer vi skriva evenutella rättelser och förtydliganden. Större förändringar görs även med annonsering i Studium. Google-dokumentet kan även användas för att kontakta lärare. Fyll i namn och nummer på ett breakout rum så kommer vi dit.

- Man kan även fråga oss lärare frågor på Slack, per email (sven-erik.ekstrom@it.uu.se och tom.smedsaas@it.uu.se), eller telefon 0720575820 (Sven-Erik) 0702544244 (Tom) (Skicka ett sms så hör vi av oss).

Inlämning:

- Inlämning av tentan sker genom uppladdning av Python-filerna på samma ställe där skrivningen hämtades (på samma sätt som en vanlig inlämningsuppgift).

- Tentan består av A- och B-uppgifter. A-uppgifter måste fungera (inlämnade program ska kunna köras och lösa uppgiften) för att godkännas. B-uppgifter kan ge "poäng" även om de inte löser problemet fullständigt.

- All inlämnad kod måste kunna köras direkt även på andra datorer än din.

- **Gör en zip-fil som innehåller alla dina filer och ladda upp den.** Om du inte vet hur du gör zip-filer kan du ladda upp filerna direkt (helst i en uppladdning). Det går att ladda upp filer flera gånger och då är det den senaste versionen som gäller. **Glöm inte bort att klicka "Skicka uppgift"!**

- Om det blir problem med Studium kan ni e-maila filerna till sven-erik.ekstrom@it.uu.se. Du måste ange din anmälningskod i brevet.

- **Inga inlämningar efter 19:00 kommer att accepteras!**

Regler

- Du får inte samarbeta med någon annan under tentan. Du får inte kopiera kod eller text utan måste skriva dina svar själv.
- Du får använda nätet.
- Du ska skriva dina lösningar *på anvisade platser* i filerna `m1.py`, `m2.py`, `m3.py` och `m4.py`. Du måste behålla namn på filer, klasser, metoder och funktioner. Du måste också ha exakt de parametrar som är angivna i uppgiften.
- Du får inte använda andra paket än de som redan är importerade i filerna såvida inte annat sägs i uppgiften. (Att ett paket finns importerat i betyder *inte* att det behöver användas!)
- Innan din tentamen blir godkänd kan du behöva förklara och motivera dina svar muntligt för lärare.

OBSERVERA ATT VI ÄR SKYLDIGA ATT ANMÄLA VARJE
MISSTANKE OM OTILLÅTET SAMARBETE ELLER
KOPIERING SOM MÖJLIGT FÖRSÖK TILL FUSK!

Tentamens beståndsdelar:

Tentamen är indelad i fyra sektioner var och en med uppgifter svarande mot de fyra modulerna. Det finns A-uppgifter och B-uppgifter i alla sektionerna.

Betygskrav:

3: Minst åtta A-uppgifter godkända.

4: Minst åtta A-uppgifter godkända samt antingen två B-uppgifter stor sett korrekt eller den frivilliga inlämningsuppgiften.

5: Minst åtta A-uppgifter godkända samt antingen alla B-uppgifter eller tre B-uppgifter och den frivilliga inlämningsuppgiften.

Uppgifter i anslutning till modul 1

Lösningen till dessa uppgifter ska skrivas på anvisade platser i filen `m1.py`. Filen innehåller också en `main`-funktion demonstrerar kodens funktion.

A1: Skriv funktionen `length_longest(lst)` som returnerar längden på den längsta av de ingående listorna (inklusive `lst`). Funktionen ska hantera sublistor med *rekursion*.

Exempel:
`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: Nedanstående funktion ordnar om listan som den får som parameter.

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

(Detta är en känd men inte särskilt effektiv sorteringsmetod och denna implementation är inte heller den bästa.)

Hur beror tiden på längden n av listan. Svara med ett Θ-uttryck.

Du kan antingen motivera svaret med ett teoretiskt resonemang utifrån vad koden gör eller genom systematisk tidmätning. Skriv svaret på anvisad plats i slutet av filen. Om du motiverar med tidmätningar så måste du skriva koden du har använt i `main`-funktionen samt de mätresultat du använder för att motivera svaret.

B1: Hur beror tiden för nedanstående funktion av parametern `n`?

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

Svara med ett Θ-uttryck. Du måste styrka ditt svar med tidmätning på körningar.

Skriv koden du använder för detta i `main`-funktionen och lägg utskrifterna på anvisad plats i slutet av koden.

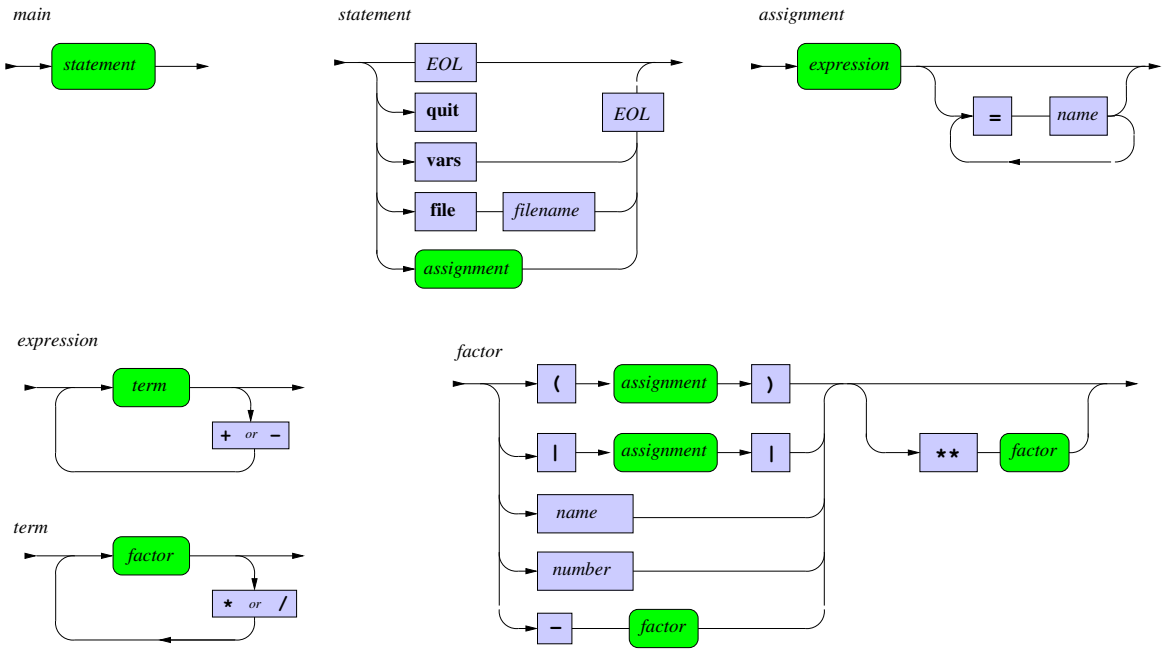
Uppskatta, med motivering, hur lång tid anropet `foo(1000000)` skulle ta.

Observera: Det ska gå att köra denna kod på annan dator!

Uppgifter i anslutning till modul 2

Den nedladdade koden `m2.py` implementerar en kalkylator liknande den i den andra obligatoriska uppgiften. Några saker är borttagna och några ska läggas till. Observera att filen också innehåller klassen `TokenizeWrapper`.

Syntaxen för uttrycken beskrivs av följande diagram:



I den givna koden finns alla funktioner (gröna rutor) men alla fungerar inte helt som de ska. Uppgifterna består alltså av att komplettera eller modifiera funktionerna och, eventuellt, lägga till några hjälpfunktioner.

Kommandot `file` utan parameter läser från filen `test.txt` (som du laddade ner). Filen börjar så här:

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

Det förväntade resultatet står efter kommentartecknet `#`.

I resten av `test.txt` ligger testfall för uppgifterna som alltså inte fungerar på rätt sätt innan du löst uppgifterna.

A3: För in vertikalstrecken `||` för absolutbelopp. Exempel:

```
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'
```

Syntaxen framgår av syntaxdiagrammen.

A4: För in operatorn `**` för ”upphöjt till” .

Exempel:

```
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
```

Syntaxdiagrammen visar var den ska implementeras. (Notera att uttrycket `x**y**z` evalueras från höger till vänster dvs som `x**(y**z)` vilket också är det vanligaste sättet i programmeringsspråk.)

Uttrycket x^y (dvs `x**y`) ska ge *evalueringsfel* om $x = 0$ och $y \leq 0$ samt om $x < 0$ och y inte har ett heltalsvärde.

Tips: Tokenizern returnerar sekvensen `122**3100` som tre tokens: `'122'`, `'**'` och `'3100'`.

B2: I den givna implementation kommer sidoeffekter som tilldelningar utföras även om det upptäcks ett syntax- eller evalueringsfel längre fram i uttrycket.

Exempel:

```
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
```

Modifiera programmet så att inga variabeltilldelningar utförs om inte uttrycket i sin helhet är korrekt och kan beräknas.

En körning med samma indata som ovan ska alltså ha utseendet

```
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: 1.0
Input : (3=x)/(12-3*4)
*** Evaluation error:  Division by zero
Input : x
Result: 1.0
```

Det finns några type av evalueringsfel som är svåra att förutse. Exempel:

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

Du behöver inte kunna hantera sådana.

I denna uppgift får du lägga till parametrar till funktionerna.

Uppgifter i anslutning till modul 3

I detta avsnitt ska du arbeta med filen `m3.py` som innehåller klasserna `LinkedList`, `BST` och `ExamException`.

Det finns också en `main`-funktion med några demonstrationskörningar. Detta är inte heltäckande tester utan du bör själv skriva fler i `main`!

Klassen `LinkedList.py` innehåller kod för att hantera *länkade listor* av objekt. Listorna är *inte* sorterade och samma värde kan förekomma flera gånger. I exemplen används heltal som data men koden ska fungera för alla typer av objekt.

Klassen `BST` innehåller kod för vanliga binära sökträd.

Klassen `ExamException` ska användas för hantering av felaktiga anrop.

A5: I den givna koden finns en metod `append` som ska lägga in ett nytt element sist i listan:

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

Metoden använder sig alltså av hjälpmetoden `_append`. Skriv klart hjälpmetoden! Den ska använda sig av *rekursion* och ska alltså inte innehåll någon iteration.

A6: Objekt ut klassen `LinkedList` kan byggas upp av en lista eller annan itererbar struktur som skickas som parameter till `__init__`-metoden:

```
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
```

Ett alternativ till rad 11 - 17 kan vara följande kod:

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

Båda versionerna finns i den nedladdade koden dvs metoden `__init__` finns i två versioner.

Hur beror körtiden i respektive fall av längden på listan? Svara med ett $\Theta(f(n))$-uttryck där n är längden på listan. Skriv svaret med motivering på anvisad plats i slutet på filen.

A7: Den givna koden implementerar metoden `__getitem__` som gör det möjligt att *hämta* ett element på ett visst index med `[ ]`-operatorn.

Implementera metoden `__setitem__` som gör det möjligt att *ändra* ett lagrat värde på ett angivet index.

Exempel:

```
    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)
```

Utskrift:

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

A8: Implementera metoden `is_omorph(self, other)` i klassen `BST` så att den returnerar `True` om det egna trädet och det andra trädet har exakt samma form oavsett innehållet i noderna, annars `False`.

Exempel:

```
    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)}')
```

Utskrift:

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

B3: Metoden `merge` i klassen `BST` infogar nycklarna från ett annat BST i det egna sökträdet:

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

För att studera det resulterande trädet kördes följande kod:

```
    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}')
```

I loopen skapas först två träd t_1 och t_2 med m respektive n nycklar. Därefter infogas de n nycklarna från t_2 till t_1. Det resulterande trädet (som kallas t_3 i utskrifterna) innehåller alltså $n + m$ nycklar.

För jämförelse skapas sedan direkt ett träd med t_4 också med $n + m$ nycklar.

Detta upprepas för en följd av värden på n och den genomsnittliga nodhöjden (dvs den interna väglängden dividerat med antalet noder) listas.

Så här ser en körning ut:

```
            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
```

Den genomsnittliga nodhöjden växer alltså kraftigt för det sammanfogade trädet t_3. Förklara varför detta blir så mycket sämre än för trädet t_4 som har lika många nycklar.

Skriv om `merge` så att det resulterande trädet t_3 inte blir sämre än t_4. Demonstrera det genom att köra koden!

Anmärkning: Om körningen tar orimligt lång tid kan du stanna på ett lägre värde på n men det måste ändå visa på att din lösning är väsentligt bättre än originalet!

Uppgifter i anslutning till modul 4

A9: En vanlig typ av matris, använd för numeriska beräkningar, har formen

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

Här är M en funktion av $\mathbf{x}$, som är en vektor (lista) av längd m, $\mathbf{x} = [x_0, x_1, x_2, \ldots, x_{m-1}]$, och $n \geq 1$ är ett heltal.

Om vi till exempel har $\mathbf{x} = [x_0, x_1, x_2] = [5, 1.5, 3]$ skulle vi få

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

och

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

Modifiera funktionen `matrix(x,n)` i `m4.py` så att den returnerar en lista, vars element själva är listor med innehållet för varje rad i matrisen $M(\mathbf{x}, n)$. **Detta ska göras med en en-rads "list comprehension".**

För de två ovantående exmplen ska du returnera följande:

```
>>> 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]]
```

Du får inte importera några extra paket eller moduler för att lösa uppgiften.

A10: Funktionen `dice(n)` i filen `m4.py` simulerar en "trasig" tärning som istället för att ha sidorna $\{1, 2, 3, 4, 5, 6\}$ har sidorna $\{1, 2, 3, 4, 5, 5\}$. Argumentet `n` ger hur många gånger tärningen kastas, och funktionen returnerar en lista med `n` slumpvisa värden som tärningen visade vid varje kast. **Du ska inte modifiera denna funktion.**

Modifiera funktionen `dice_average()` i filen `m4.py` så att den gör följande:

- Kör funktionen `dice(n)` med $n = 100000$, tjugo gånger parallellt, d.v.s. totalt 2000000 "kast". Du kan välja vilket parallelliserings-paket eller modul du vill, det viktiga är inte att koden blir effektivare.
- Returnera (d.v.s. inte printa) medelvärdet för alla kast (d.v.s. bara ett värde).

Du får inte importera några extra paket eller moduler för att lösa uppgiften.

B4: Du ska skriva ett program som skapar thumbnails (nerskalad storlek) av alla bild-filer som finns i samma katalog som Python-programmet som körs. I denna uppgift får ni använda vilka paket eller moduler ni vill. Se till att läsa tipsen i slutet på denna uppgift.

Modifiera funktionen `thumbnail()` i `m4.py`, och det är tillåtet att skriva andra hjälp-funktioner.

Programmet ska

- fungera både på `.png` och `.jpg` filer automatiskt;
- inte ha några inparametrar (d.v.s. det ska hitta alla bild-filer själv);
- generera thumbnails med samma ratio mellan antalet pixlar på bredd och höjd som original-bilden, men max av dessa ska vara 100 pixlar;
- spara thumbnails i en katalog med namn `thumb`. Om katalogen inte finns när programmet körs, så ska katalogen skapas. Lägg till `thumb_` till original-filnamnet, t.ex. `fish.png` blir `thumb_fish.png`;
- ej ha hårdkodade "path" var filerna ligger på er privata hårddisk;
- exekveras parallellt (d.v.s. skapandet av thumbnails sker parallellt).

Tips:

- Bland de nerladdade filerna finns tre exempel-bilder ni kan testa på (`giraffe.jpg`, `moose.jpg`, och `fish.png`). Programmet ska dock fungera med godtyckliga namn och antal.
- Paketet `pillow` kan manipulera bilder, och vi rekommenderar att ni använder detta. Efter installation av `pillow` importerar ni den med: `from PIL import Image` (Ni får använda andra paket än denna om ni vill det).
- I paketet `os` kan ni hitta funktioner för att hitta namn på filer i en katalog m.m.
- Börja med att få koden att fungera utan parallellisering.