Tentamen på programmeringsteknik II 2021-08-26

Skrivtid: 14:00 – 19:00

Observera: Denna skrivning är avsedd för de studenter som läst kursen ht 2020 eller senare. Skrivningen för de som läst kursen tidigare med Java ska gå till hit (or mail tom.smedsaas@it.uu.se in case of trouble).

Praktiska detaljer och hjälp under tentamen:

- Under stor del av tentan kommer lärare att finnas tillgängliga i detta Zoom-rum. 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 detta dokument öppet under tentans gång. Där kommer vi skriva eventella rättelser och förtydliganden. 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 att man laddar upp filer som en vanlig inlämningsuppgift, kallad **Tentamen** i Studium.

- Filerna som ska laddas upp heter `m1.py`, `m2.py`, `m3.py`, och `m4.py` och ”skalen” för dessa, som ni ska modifiera, kan ni ladda ner under uppgiften **Tentamen** i Studium.

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

- Vi ser helst att ni laddar upp filerna till Studium i en enda uppladdning (”drag-and-dropp:a” alla filer på en gång till ”Ladda upp fil”) under uppgiften **Tentamen**. **Glöm inte bort att klicka ”Skicka uppgift”!** En zip-fil går också bra. Att ladda upp en fil i taget är också OK.

- Ni kan ladda upp filer flera gånger under tenta-tiden genom att klicka ”Försök igen”, då är det sista versionen som laddats upp som räknas som den ni lämnat in.

- Om det blir problem med Studium kan ni e-maila filerna till sven-erik.ekstrom@it.uu.se.

- **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`. Skriv ditt namn och din kod (om du fått någon) i kommentarerna i början av varje fil. Du måste behålla namn på filer, klasser, metoder och funktioner.
- 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:

I tentamen finns A-uppgifter och B-uppgifter.

Tentamen är indelad i fyra sektioner var och en med uppgifter svarande mot de fyra modulerna.

Till modul 1 hör uppgifterna:

A1: Använd rekursion för att räkna förekomster av ett angivet värde.
A2: Utnyttja *memorering* (eng. *memoization*) för att effektivisera en given funktion.
B1: Tidsuppskattning för en rekursiv funktion.

Till modul 2 hör uppgifterna:

A3: En operator för restberäkning.
A4: Ett `clear`-kommando för att tömma variabellistan.

Till modul 3 hör uppgifterna:

A5: En metod för att ta bort alla element med ett visst värde ur en länkad lista.
A6: En komplexitetsberäkning för upprepade inlägg i listan.
A7: En metod för att räkna löv i ett binärt sökträd.
A8: Överlagring av operatorn `==` för binära sökträd.
B2: En inläggningsmetod för länkade listor.
B3: Komplexitetsanalys av en given funktion.

Till modul 4 hör uppgifterna:

A9: Parallellisering.
A10: Förstå och modifiera kod.
B4: Högre ordningens funktioner.

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 som testar kod-uppgifterna.

A1: Skriv funktionen `count_all(lst, d)` som räknar hur många gånger `d` finns på *alla* nivåer i `lst` Funktionen ska hantera sublistor med *rekursion*.

Exempel:
`count_all([], 1)` = 0
`count_all([1], 1)` = 1
`count_all([1], 0)` = 0
`count_all([1,2,1,3,[[1],[1,2,3]]], 1)` = 4

Du kan förutsätta att det första argumentet `lst` är en lista (eventuellt tom). Det är tillåtet att iterera över listan men sublistor måste alltså behandlas rekursivt. Det är alltså inte tillåtet att ”platta ut” listan.

A2: Nedanstående funktion kan, i denna utformning, inte hantera särskilt stora värden på argumentet n. Skriv funktionen `c_mem(n)` som fortfarande använder *rekursiva* anrop men också också tekniken med *memorering* (även kallad *dynamisk programmering* eller, på engelska, *memoization*) för att klara stora värden på n. Beräkna `c(100)` och ange värdet.

```
1 def c(n):
2     if n <= 2 :
3         return 1
4     else:
5         return c(n-1) - c(n-3)
```

B1: Hur lång tid skulle det ta att beräkna `c(100)` på din dator med koden i uppgift A2 oförändrad. Motivera svaret!

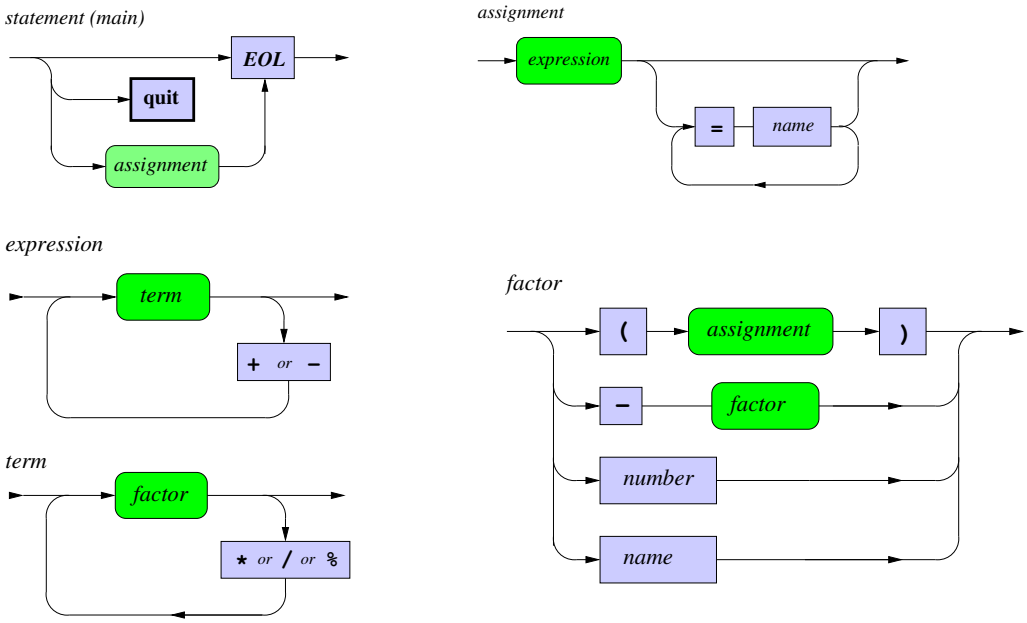
Eftersom du ska svara för vilken tid det skulle ta på din dator så behöver du säkert göra vissa tidmätningar. Skriv koden du använder för detta i `main`-funktionen. Redovisa ditt resonemang i ”flerradssträngen” i slutet av koden.

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.

Programmet börjar med att läsa kommandon från en fil med namnet `init.txt` (som du laddade ner). Filen börjar så här:

```
(2+4=x) - x                          # 0.0
2*(3*x + 2)/(16-x)                   # 4.0
```

Det förväntade resultatet står efter kommentartecknet `#`.

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

A3: Lägg till ”restoperatorn” `%`. Exempel:

```
Input : 1%2
Result:  1.0
Input : 3%3
Result:  0.0
Input : 10 + 23%17 + 2
Result:  18.0
Input : 4%0
EvaluationError:  Division of 4.0 by zero
Input :  3%4%2
Result:  1.0
Input : 5%(3*2)
Result:  5.0
Input : (5+3)%(2*3)/2
Result:  1.0
Input : (5+3)%(2*3/2)
Result:  2.0
Input : (5+3)%(2*3/2 - 1.5*2)
EvaluationError:  Division of 8.0 by zero
```

Operatorn ska alltså ha *samma* prioritet som `*` och `/`. Vid lika prioritet utförs operationerna från vänster till höger.

A4: Lägg till `clear`-kommandot `main` som tar bort alla definierade variabler.

Exempel:

```
Input : 1 = x = y = z
Result:  1.0
Input : z + x
Result:  2.0
Input : clear
Input : y
EvaluationError:  Undefined variable: y
```

Uppgifter i anslutning till modul 3

I detta avsnitt ska du arbeta med filen `m3.py` som innehåller klasserna `LinkedList.py` och `BST.py` samt funktionen `bst_sort`.

Det finns också en `main`-funktion med några enkla tester.

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.

A5: I den givna koden för `LinkedList` finns en metod

```
    def remove_all(self, x):
        self.first = self._remove_all(x, self.first)
```

som ska ta bort alla noder som innehåller `x` ur listan.
Skriv den *rekursiva* hjälpmetoden `_remove_all(self, x, f)` som tar bort alla noder som innehåller `x` ur listan som börjar med nod `f` och returnerar den första noden i den resterande listan.

A6: Hur beror körtiden för nedanstående kod av n?

```
lst = LinkedList()
for x in range(n):
    lst.add_last(x)
```

Svara med ett Θ-uttryck och motivera! Skriv svaret i textsträngen i slutet på den nedladdade filen!

A7: I klassen `BST` finns metoden `count_leaves(self)` som ska returnera antalet löv i trädet (dvs antalet noder som saknar barn). Metoden använder hjälpmetoden `_count_leaves(self, r)`. Skriv klart hjälpmetoden!

A8: Överlagra operatorn `==` i klassen `BST` så att den returnerar `True` om två träd innehåller samma data (oavsett form), annars `False`.

Exempel:

Kod:

```
    bst1 = BST([2, 10, 1, 7, 5, 8, 3])
    bst2 = BST([10, 1, 5, 3, 7, 8, 2])
    print(bst1 == bst2)
    bst1.add(4)
    print(bst1 == bst2)
    print(BST()== BST())
    print(BST()== BST(1))
```

Utskrift:

```
True
False
True
False
```

B2: Skriv en metod `insert(self, x, index=0)` i klassen `LinkedList` som lägger en ny nod med `x` som data på position `index` i listan.

Exempel (kommentarerna visar resulterande lista):

```
lst = LinkedList()
lst.insert(3)           # <3>
lst.insert(5, 1)        # <3, 5>
lst.insert(5)           # <5, 3, 5>
lst.insert(4,1)         # <5, 4, 3, 5>
lst.insert(1, 99)       # Index out range: 99
```

B3: Hur beror tiden för nedanstående funktion av längden n på listan `aList` (standard Python-lista) som kan förutsättas innehålla slumptal?

```
def bst_sort(aList):
    bst = BST()
    for x in aList:
        bst.add(x)
    result = []
    for x in bst:
        result.append(x)
    return result
```

Svara med Θ- eller (realistiska) $\mathcal{O}$-uttryck.

Motivera svaret antingen teoretiskt eller med redovisade experiment eller båda delarna.

Skriv svaret i avsedd textsträng i slutet på filen!

Uppgifter i anslutning till modul 4

A9: Filen `customer.json` (som finns bland de nerladdade filerna) är en så kallad JSON-fil (ett textbaserat filformat för att lagra data av olika slag).

Denna fil innehåller olika kunddata för ett fiktivt företag; ett antal personer som alla har ett index (0,1,2,...) med olika egenskaper.

I filen `m4.py` finns funktionen `get_balance(index)` som använde modulen `json` för att extrahera fältet `balance` (saldo) ur `customer.json` för personen med givet index (0,1,2,...,111), då det finns 112 kunder. Om man anropar `get_balance(0)` får man alltså första personens (index 0) `balance` som är 1801.02. Om man öppnar filen `customer.json` i en editor ser man att för person 0 är

```
"balance": "$1,801.02"
```

men funktionen rensar automatiskt bort `$` och `,`.

Skriv metoden `get_total_balance()` i `m4.py` som returnerar det totala (summan) av alla kunders 'balance' (saldo). Hämtningen av alla personers `balance` ska ske parallellt. Ni kan använda vilken parallellisering ni vill för att lösa detta, målet är inte snabbare kod utan att visa att ni kan parallellisera kod.

A10: Ni ska i denna uppgift återigen använda filen `customer.json`, som ni använde i uppgift A9.

Modifiera metoden `get_mean_balances()` så att den returnerar medel-saldo ('balance') för manliga och kvinnliga kunder. Varje kund har ett fält 'gender', som antingen är 'male' eller 'female'. Det spelar ingen roll om ni returnerar en lista eller tuple.

B4: Ett skottår (ett år då man lägger till en extra dag) är definierat som

Varje år som är jämnt delbara med fyra är ett skottår, förutom år som är jämnt delbara med 100, men de sekelår som är jämnt delbara med 400 är skottår. Detta innebär till exempel att åren 1700, 1800 och 1900 inte är skottår, men åren 1600 och 2000 är det.

Modifiera metoden `leapyears(years)`, som har argumentet 'years' som antas vara en lista av år, så den returnerar alla skottår i listan. Använd en enrads 'comprehension' som skapar en lista av alla skottår.