Regler

- Student får lämna salen tidigast en timme efter tentans start.
- Vid toalettbesök eller rökpaus ska pauslista utanför salen fyllas i.
- All form av kontakt mellan studenter under tentans gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av tentavakt i samband med tentans start samt under tentans gång.
- Frågor om specifika uppgifter eller om tentan i stort ska ställas via tentasystemet.
- Systemfrågor kan ställas till assistent i sal.
- Ingen uppgift rättas efter tentatidens slut.
- Ingen kompletteringsmöjlighet ges de sista tio minuterna.
- Så länge en uppgift inte har betyget U kan den kompletteras till godkänt.
- Inga elektroniska hjälpmedel får medtas. Mobiltelefon ska vara avstängd och ligga i jacka eller väska.
- Inga ytterkläder eller väskor får förvaras vid skrivplatsen.

Antal uppgifter	5
Antal sidor (exklusive denna)	5
Hjälpmedel	En pythonbok (tex Learning Python 5 ed.) En A4-sida med egna anteckningar

VÄND!

Betygssättning

Tentan innehåller fem uppgifter. För godkänt betyg krävs minst två avklarade uppgifter. För gränser för högre betyg, se i tabell 1.

Tid (timmar)	Antal uppgifter	Betyg
3	3	5
4	4	5
4	3	4

Tabell 1: Betygssättning

Bonus från labserien

Bonus från labserien adderas på respektive tidsgräns för högre betyg. Varje avklarad deadline under kursens gång ger fem minuters extra tid. Observera att bonusen endast gäller under detta tillfälle och endast gäller gränserna för betyg 4 och 5. Tentamenstiden slutar 13.00 oavsätt bonustid.

Information

Inloggning

Välj session “exam system” och logga in med din vanliga IDA-inloggning (vanligtvis liu-id som användarnamn och samma lösenord som på studentportalen).

Följ menyvalen så långt det går tills du ska mata in ett engångslösenord. Tag fram ditt LiU-kort och visa det för tentavakten för att få detta lösenord.

Avslutning

Logga ut från systemet som vanligt via meny, vänta ett tag och tryck sedan på knappen “Avsluta tentamen” när det är möjligt. När detta är gjort är det omöjligt att logga in igen.

Uppgift 1

Du jobbar för skatteverket och har glömt vart programmet som skapar personnummer finns. Då du har en lång kö av bebisar som behöver ett personnummer måste du snabbt skapa ett nytt program som genererar personnummer.

Programmet ska be användaren om ett födelsedatum och kön på personen som ska få ett personnummer och sedan skriva ut ett giltigt personnummer. De två första siffrorna efter datumet ska slumpas fram. Näst sista siffran ska vara jämn för kvinnor och udda för män men slumpmässig i övrigt och sista siffran ska beräknas enligt Luhn-algoritmen.

Enligt Luhn-algoritmen (även kallad mod-10-algoritmen) multipliceras siffrorna växelvis med 2 och 1 och därefter summeras produkterna. För en tvåsiffrig produkt beräknas summan av dess siffror. Summan dras sedan bort från närmsta övre tiotal för att få fram kontrollsiffran.

För att beräkna kontrollsiffran för personnumret `811218-987` utförs följande uppställning:

```
   8 1 1 2 1 8 - 9 8 7
*  2 1 2 1 2 1   2 1 2
-----------------------
   ^ ^ ^ ^ ^ ^   ^ ^ ^
  16 1 2 2 2 8  18 8 14
```

$$1 + 6 + 1 + 2 + 2 + 2 + 8 + 1 + 8 + 8 + 1 + 4 = 44$$

Kontrollsiffran är $50 - 44 = 6$

Du behöver inte här ta hänsyn till att personnummret redan genererats (vilket så klart måste göras på skatteverket) men ditt program ska normalt sett ge ett nytt personnummer vid varje körning. Inmatning och utskrift ska ske enligt körexemplen nedan. Du behöver inte kontrollera användarens inmatning.

Körexempel 1

```
Födelsedatum: 910323
Kön (M/F): F

910323-3764
```

Körexempel 2

```
Födelsedatum: 781012
Kön (M/F): M

781012-6172
```

Uppgift 2

I kortspelet black jack är målet att ta kort och komma så nära summan 21 som möjligt utan att gå över. Alla klädda kort (knekt, dam och kung) är värda 10, ess är värda ett eller elva (valfritt) och övriga kort har sina numeriska värden (2 till 10). I modulen `card.py` finns en ADT för att hantera ett kort och modulen `deck.py` innehåller en ADT för att hantera en kortlek. Använd dessa två moduler för att skapa ett program som till att börja med ger användaren två kort och skriver ut totalvärdet av användarens hand. Därefter ska användaren få välja om hen ska ta fler kort eller stanna så länge totalsumman är mindre än 21. Om det finns med ett eller flera ess ska samtliga alternativ med värde under 22 skrivas ut.

OBS! Du får inte ändra i den givna koden. Du ***ska*** även använda dig av de primitiver som finns givna, det är inte säkert att vi använder samma datastruktur för att representera kort eller kortlek när vi testar er kod. Däremot har våra ADTer så klart samma primitiver. Du behöver inte skicka in den givna koden vid rättning.

I följande körexempel har kortsymbolerna bytts ut mot tecknen h, c, s, d (för hjärter, klöver, spader respektive ruter).

Körexempel 1

```
Dina kort:  h2  dK
Summa: 12
Vad vill du göra? (s för stanna, t för ta ett till): t
Dina kort:  h2  dK  sA
Summa: 13
Vad vill du göra? (s för stanna, t för ta ett till): t
Dina kort:  h2  dK  sA  c4
Summa: 17
Vad vill du göra? (s för stanna, t för ta ett till): s
```

Körexempel 2

```
Dina kort:  c6  hA
Summa: 7 / 17
Vad vill du göra? (s för stanna, t för ta ett till): t
Dina kort:  c6  hA  dQ
Summa: 17
Vad vill du göra? (s för stanna, t för ta ett till): t
Dina kort:  c6  hA  dQ  cK
Summa: 27
```

Uppgift 3

Som ni alla vet är filsystemet i UNIX (och Linux) byggt i en trädliknanden struktur där '/' finns överst och sedan separeras kataloger i en sökväg med tecknet '/'. Man kan även ange den relativa sökvägen '`..`' som betyder ett steg uppåt. Man kan aldrig gå högre upp än '/', detta leder till att sökvägen '`/../`' är felaktig.

Ibland kan man dock tycka att det blir lite jobbigt att skriva sökvägar på detta sett och därför ska du nu skapa ett program som hanterar sökvägar som skrivs på ett litet annorlunda skrivsätt. Kataloger ska separeras med tecknet större än (>) och tecknet mindre än (<) ska motsvara katalogen ett steg ovanför (`../`). Du behöver i denna uppgift endast hantera absoluta sökvägar (utgående ifrån >) Nedanstående tabell visar exempel på sökvägar i de olika systemen:

Sökväg	Normaliserad sökväg
`>`	`/`
`>usr>bin`	`/usr/bin`
`>usr>bin<lib`	`/usr/lib`
`><`	`/`
`><<<`	`/`
`><TDP002`	`/TDP002`
`>TDP002>www-pub<<<TDP001`	`/TDP001`
`>home>TDP002>www-pub<<TDP001`	`/home/TDP001`

Körexempel 1

```
Mata in sökväg: >home>TDP002>www-pub
Normaliserad: /home/TDP002/www-pub
```

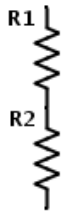
Körexempel 2

```
Mata in sökväg: >home>TDP002>www-pub<<TDP001
Normaliserad: /home/TDP001
```

Du kan anta att inga sökvägar har flera >-tecken i rad. Sökvägar innehåller inte heller tecknen '`.`' eller '/'.

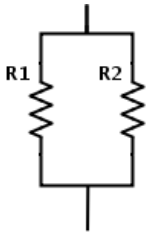
Uppgift 4

Ett motstånd är en elektronisk komponent som ger en viss resistans (som mäts i ohm, Ω) i en krets. Motstånd kan kombineras genom att seriekoppla (figur 1) eller parallellkoppla (figur 2) för att uppnå den resistans man är ute efter.



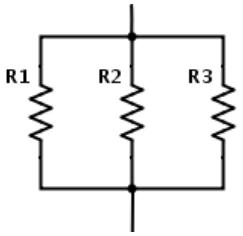
$$R_{tot} = R_1 + R_2$$

Figur 1: Seriekopplade motstånd



$$R_{tot} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

Figur 2: Parallellkoppling



$$R_{tot} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

Figur 3: Parallellkoppling

I denna uppgift ska du ***REKURSIVT*** beräkna hur många 100Ω-motstånd som behövs för att uppnå en given resistans enligt algoritmen nedan. Observera att du alltid kommer sluta med likadana kretsar av parallellkopplade motstånd kopplade i serie.
Med antagandet att vi alltid har motstånd om 100Ω kan vi förenkla formlerna i figur 2 och 3 till nedanstående formel med N motstånd:

$$R_{tot} = \frac{1}{\frac{N}{100}} = \frac{100}{N}$$

Algoritm:

```
Steg 1:
Börja med ett krets innehållandes ett motstånd med resistans 100

Steg 2:
Om kretsen kan kopplas i serie flera gånger för att exakt nå målresistansen:
  Multiplicera antalet motstånd i kretsen med antal kretsar som seriekopplas
Annars:
  Lägg till ett till motstånd med 100 ohm parallellt med övriga i kretsen.
  Gå vidare med den nya kretsen i steg 2.
```

Körexempel 1

```
Mata in målresistans: 250
Det behövs 10 motstånd
```

Körexempel 2

```
Mata in målresistans: 17
Det behövs 1700 motstånd
```

KRAV: Du ska använda dig av den givna algoritmen.

Uppgift 5

Du ska i denna uppgift skapa ett program som slår samman innehållet i två filer på ett speciellt sätt. Programmet ska ersätta en rektangel av text från en fil med en lika stor rektangel från en annan fil. Resultatet ska skrivas ut i terminalfönstret. Inga av indatafilerna ska förändras. Vid programstart ska programmet ges fyra kommandoradsargument beskrivna nedan:

Originalfilen (O) Filen vars innehåll ska ersättas

Utbytestexten (R) Filen där utbytestexten tas ifrån

Starttecken (S) Efter hur många tecken på varje rad i O som rektangeln ska starta

Rektangelns bredd (B) Hur många tecken på varje rad som ska bytas ut

Antalet rader i rektangeln är alltid samma som antal rader i R. Alla rader ska efter bytet vara minst S+B tecken långa. Vid behov ska rader fyllas ut med mellanslag för att få nog stor bredd. Du kan anta att originalfilen alltid har minst lika många rader som utbytesfilen. Det kan dock finnas både rader som är kortare och längre än B.

I mappen `given_files/text/` finns det några textfiler att testa med.

Körexempel 1

```
$python3 replace_rectangle.py given_files/text/O1 given_files/text/R1 4 5
****-----**********
****-----**********
****-----**********
**********************
**********************
**********************
```

Körexempel 2

```
$python3 replace_rectangle.py given_files/text/O2 given_files/text/R2 6 6
Detta blir  lite mer
normalt (?) med
vacker text
```

Körexempel 3

```
$python3 replace_rectangle.py given_files/text/O1 given_files/text/R3 2 3
**-  **************
**/& **************
**///**************
**(((**************
**...**************
**********************
```