Programmering og Problemløsning (PoP)
Reeksamen, Block 3, 2020

Underviserne på PoP

15. april, 2020

version 1.0

Formalia

Dette dokument udgør reeksamensopgaven for kurset Programmering og Problemløsning (PoP), forår 2020. Dokumentet og dets supplerende filer udleveres via Københavns Universitets digitale eksamenssystem.

Eksamensbesvarelsen afleveres via universitetets digitale eksamenssystem.[1] Opgaven er individuel og skal besvares individuelt. Bemærk at løsninger hverken må diskuteres i grupper eller med andre, og at overtrædelser vil blive betragtet som eksamenssnyd.

Der skal afleveres en rapport der besvarer de enkelte spørgsmål (en pdf-fil navngivet *KU-nummerplade*.pdf) samt en zip-fil (navngivet *KU-nummerplade*.zip), der indeholder programkode, tests og lignende (gerne i kørbar stand). Omfanget af rapporten er 11-15 normalsider. En normalside er 2400 tegn (inkl. mellemrum). Bemærk, at for særlig teknisk tekst (herunder programtekst) er en normalside 1200 tegn (inkl. mellemrum). Rapporten skal som udgangspunkt kunne læses isoleret fra programkoden omend rapporten gerne må referere til programkoden, f.eks. som dokumentation for kørte tests og for at de skrevne F# funktioner virker i en kontekst.

Der gælder følgende:

- Rapporten skal udarbejdes på dansk, engelsk, norsk eller svensk. Der skal anvendes samme sprog gennem hele rapporten (undtaget herfra er evt. variabel- eller funktionsnavne anvendt i programteksten) samt tekst genereret af programdelene.
- Rapporten skal indeholde en forside med navn og KU-nummerplade (f.eks. "xyz123").
- Der skal benyttes kort, præcist, klart og korrekt sprog.

[1] **Bemærk:** I tilfælde af tekniske problemer med universitetets digitale eksamenssystem kan besvarelsen i nødstilfælde sendes direkte til de ansvarlige undervisere for reeksamen (mael@di.ku.dk og c.lioma@di.ku.dk) med cc til uddannelse@diku.dk. Husk at inkludér et screen-dump af problemet. Afleveringsfristen (se ovenfor) står ved magt i sådanne tilfælde.

- Afleveringen skal indeholde en litteraturliste efter behov. Som I alle akademiske opgaver skal kilder angives i det omfang materiale (f.eks. F# funktioner) benyttes fra undervisningsmateriale eller fra andre kilder.

- Zip-filen skal indeholde to biblioteker navngivet, henholdsvis, "deathrow" og "pandemic", der har til formål at indeholde programfiler for besvarelserne på de to delopgaver eksamensopgaven består af.

Vurdering af besvarelsen

Besvarelsen bedømmes som bestået eller ikke-bestået. Bedømmelsen vil fremgå af universitetets eksamensprotokoller senest 4 uger efter afleveringsfristen.

Hvert spørgsmål tilknyttes en procentsats, således at procentsatserne for alle spørgsmålene summer op til 100 procent. Vurderingen af besvarelserne på hvert spørgsmål vil bero på et helhedsindtryk. Helhedsindtrykket tager udgangspunkt i beskrivelserne og overvejelserne beskrevet i svaret, hvor godt den enkelte besvarelse er disponeret, hvor koncis og korrekt besvarelsen fremstår og i hvilken grad problemet er løst. Herudover lægges der vægt på at programteksten fremstår overbevisende og klar.

Eksamensopgaven består af to delopgaver der vægtes lige (spørgsmålene for hver delopgave summer op til 50 procent). For at opnå bedømmelsen "bestået" skal afleveringen bedømmes til over 50 procent af en 100 procent besvarelse.

Redelighed

Universitetets normale krav om redelighed i opgavebesvarelser skal overholdes (se Regler om disciplinære foranstaltninger over for studerende ved Københavns Universitet og Københavns Universitets regler om god videnskabelig praksis).

Praktisk betyder ovenstående, at man ikke må snyde til eksamen. Eksamenssnyd kan f.eks. være at

- få bistand til at besvare sin eksamensopgave (fra andre end sine gruppemedlemmer)

- hjælpe andre til at få bistand til besvarelse af deres opgave

- forfalske eller fortie resultater

- gengive andres tanker eller materiale uden at kildehenvise

- gengive egne tanker eller materiale fra tidligere indleverede opgaver uden at kildehenvise

Hvis underviserne får mistanke om eksamenssnyd behandles sagen iht. universitetets regler ovenfor. Bemærk, at de disciplinære konsekvenser ved eksamenssnyd kan have betydelig

indvirkning på ens fremtidige studieaktivitet.[2]

Eksamensopgavens tekst (dette dokument) må ikke deles med andre. Ligeledes må potentielle løsninger til eksamensopgaven ikke diskuteres med andre personer før efter eksamensperiodens udløb. Til gengæld står det studerende frit for at diskutere fortolkninger af opgaven (hvad der skal laves) med andre studerende. Man må altså gerne diskutere hvad der skal laves, men ikke hvordan det skal laves.

Hjælp under eksamensperioden

Diskussionsforummet på kursets hjemmeside vil i tidsrummet 9-21 på eksamensdagen blive læst af en underviser, som vil svare på spørgsmål om eksamensopgaven. Kravene om redelighed gælder også på diskussionsforummet. Bemærk, at programtekst ("kode") regnes for en del af en potentiel eksamensbesvarelse og derfor ikke må postes på forummet.

Gode råd

Nedenfor er gode råd, som tidligere har vist sig lønsomme for mange studerende i deres arbejde med eksamensopgaver. Det er valgfrit, om man følger disse råd.

- Læs hele opgaveteksten, før besvarelsen påbegyndes.
- Fastlæg en arbejdsplan for eksamensperioden. Husk at selv de bedste planer skal laves om, hvis situationen ændrer sig.
- Begynd at se på opgaverne med det samme. Opgaverne er uafhængige af hinanden og det kan være fornuftigt at få en opgave lidt på afstand for at vende tilbage til den senere.
- Læs korrektur på rapporten inden den afleveres (og gerne løbende). Rapporten er produktet af din indsats—det er vigtigt at den fremstår overbevisende.
- Husk at en løsning ofte har nogle begrænsninger eller utilsigtede effekter, og at dette ikke i sig selv er et problem, så længe begrænsningerne og effekterne er beskrevet i besvarelsen, gerne med en angivelse af, hvordan de kan afhjælpes. Det kan derfor ofte være en god strategi at beskrive løsningens begrænsninger og utilsigtede effekter, frem for at bruge meget tid på at forsøge at løse dem.
- Undlad gentagelser og uklare formuleringer i besvarelsen. Læs derfor din besvarelse igennem minimum en gang for at fjerne sådanne.
- Aflever eksamensbesvarelsen! Også hvis du mener at du umuligt kan bestå.

[2]Underviserne i dette kursus udfører automatiseret plagiatkontrol på både rapporter og programtekst. Bemærk, at det er uhyre let for automatiserede metoder at afsløre plagieret programtekst, selv hvis funktioner og variable er omnavngivet og forekommer i anden rækkefølge end den oprindelige tekst.

```
type prisoner = {first_name              : string;
                 last_name               : string;
                 date_of_birth           : string;
                 date_of_offence         : string;
                 education_level         : int option;
                 execution_date          : string;
                 age_at_execution        : int;
                 date_received           : string;
                 race                    : string;
                 eye_color               : string;
                 weight                  : int option;
                 height                  : string;
                 county                  : string;
                 native_county           : string;
                 native_state            : string;
                 last_statement          : string
                 }
```

Figure 1: Typen `prisoner`.

Opgave 1: Personer på dødsgangen

I forbindelse med en reform af kriminalforsorgen i USA ønsker man at beregne statistik for dødsdømte i det håb at kunne bruge statistikken til senere forebyggende arbejde. I det følgende skal vi arbejde med et datasæt for dødsdømte i staten Texas, USA. Datasættet er udtrukket af Zi Chong Kao fra Texas Department of Criminal Justice hjemmeside. Information fra før 1995 er i nogle tilfælde kun tilgængeligt som billeder af fysiske dokumenter, og er udtrukket manuelt.

Datasættet er tilgængeligt i filen `deathrow.fs`, som kan hentes på Absalon. Filen indeholder et modul `Deathrow`, som indeholder en erklæring af en type `prisoner` (se Figur 1) og en erklæring af en variabel `deathrow` af type `prisoner list`. En værdi af typen `prisoner` beskriver en henrettelse. Følgende er de første to elementer i listen `deathrow`:

```
let deathrow : prisoner list =
  [{first_name="Christopher Anthony"; last_name="Young"; date_of_birth=
   "1983-09-24"; date_of_offence="2004-11-21"; education_level=Some 9;
   execution_date="2018-07-17"; age_at_execution=34; date_received="
   2006-03-31"; race="Black"; eye_color="Brown"; weight=Some 216;
   height="6' 1\""; county="Bexar"; native_county="Bexar"; native_state
   ="Texas"; last_statement="I want to make sure the Patel family knows
    I love them like they love me. Make sure the kids in the world know
    I'm being executed and those kids I've been mentoring keep this
   fight going. I'm good Warden."};
   {first_name="Danny Paul"; last_name="Bible"; date_of_birth="
   1951-08-28"; date_of_offence="1979-05-27"; education_level=Some 12;
   execution_date="2018-06-27"; age_at_execution=66; date_received="
```

```
  2003-07-17"; race="White"; eye_color="Blue"; weight=Some 194; height
  ="5' 7\""; county="Harris"; native_county="Brazoria"; native_state="
  Texas"; last_statement=""};
 ...]
```

Hvert objekt, der beskriver en henrettelse, har 16 felter. De fleste felter er selvbeskrivende, men følgende forklarer nogle, hvis betydninger ikke er umiddelbart åbenlyse:

- `education_level` er et heltal (optional), der angiver højeste gennemførte klassetrin. Kan være `None` hvis ukendt.
- `date_received` er dato for ankomst til dødsgangen.
- `weight` er et heltal (optional), der angiver vægt i amerikanske pund. Kan være `None` hvis ukendt
- `height` er højde i fod og tommer (feet and inches). Kan være den tomme tekst (””) hvis ukendt.

Generelt kan de fleste tekstfelter være den tomme tekst, hvis værdien for feltet er ukendt.

Spørgsmål 1.A (5 %)

Højden på fangerne er givet i fod og tommer, som en tekst på formen f`' `i`"`, hvor f og i er heltal. Altså: Et heltal efterfulgt af en apostrof (`'`), efterfulgt af mellemrum, efterfulgt af et heltal, efterfulgt af gåseøjne (`"`).

Én fod er 12 tommer, og én tomme er 2,54 cm. Fod og tommer kan omregnes til centimeter (cm) ved først at omregne fod til tommer (dvs. gange med 12), og dernæst omregne tommer til centimeter (dvs. gange med 2,54). Amerikanske pund kan omregnes til kilogram (kg) ved at dividere med 2,2046. Den amerikanske måde at vise længder i fod og tommer samt vægt i pund kaldes imperial, mens at den danske måde der bruger centimeter og kilogram kaldes metric.

Konstruer en F# funktion `imperial_to_kg` af type `int`→`int`, som omregner en imperial vægtstørrelse i pund til kilogram. Funktionen skal afrunde til det nærmeste heltal.

Opstil en række tests for funktionen og sammenlign en afprøvning af din funktion med de forventede resultater.

Spørgsmål 1.B (10 %)

Konstruer en F# funktion `imperial_to_cm` af type `string`→`int option`, som omregner en imperial højde (angivet i fod og tommer) til et antal centimeter. Funktionen skal returnere `None` hvis inputstrengen ikke matcher specifikationen ovenfor. Som eksempel skal`imperial_to_cm "6' 2 \\""` returnere værdien `Some 187.96`.

Opstil en række tests for funktionen og sammenlign en afprøvning af din funktion med de forventede resultater.

Spørgsmål 1.C (10 %; afhænger af 1.B)

Konstruer en F# funktion `mean_height`, af type `prisoner list` → `int option`, der returnerer gennemsnitshøjden, hvis en sådan findes, i cm (afrundet) for en liste af fanger. Giv også gennemsnitshøjden for de af fangerne i listen `deathrow`, der er angivet med en højde.

Giv både en imperativ og en funktionel implementation af funktionen. Den funktionelle version må meget gerne gøre brug af funktionen `foldl`.

Opstil en række tests for funktionen og sammenlign en afprøvning af din funktion med de forventede resultater.

Spørgsmål 1.D (10 %; afhænger af 1.A og 1.B)

Konstruer en F# funktion `pr_prisoners`, af type `prisoner list→unit`, der udskriver, som text på stdout, en tabel over fangerne i listen. Tabellen skal have fire kolonner, fangens fulde navn, fangens alder ved henrettelse, højde (i cm) og vægt (i kilogram). Tabellen skal udskrives med en række pr fange og sorteret efter alder ved henrettelsen (yngste først).[3]

Ved at kalde funktionen på et datasæt indeholdende de fanger der blev henrettet i 2018 kunne man evt få udskrevet følgende tabel:

```
| Name                       |         Age |      Height |      Weight |
| Erick Daniel Davila        |          31 |       180cm |        73kg |
| Christopher Anthony Young  |          34 |       185cm |        97kg |
| Juan Edward Castillo       |          37 |       180cm |        81kg |
| Rosendo Rodriguez III      |          38 |       173cm |        89kg |
| Anthony Shore              |          55 |       178cm |        89kg |
| John David Battaglia       |          62 |       183cm |        85kg |
| William Earl Rayford       |          64 |       193cm |        81kg |
| Danny Paul Bible           |          66 |       170cm |        87kg |
```

Spørgsmål 1.E (10 %)

Konstruer en F# funktion `county_stats`, af type `prisoner list→(string*int)list`, som tager en liste af `prisoner`-værdier som argument og returnerer statistik over hvormange henrettelser der er foretaget i hver county, ifølge argumentlisten.

Opstil en række tests for funktionen og sammenlign en afprøvning af din funktion med de forventede resultater.

Spørgsmål 1.F (5 %)

Konstruer en F# funktion `find_text`, af type `string→prisoner list→prisoner list`, som tager en streng og en liste af `prisoner`-værdier som argumenter og returnerer de fanger i input-

[3]Du kan enten benytte dig af F#'s indbyggede `List.sortBy` funktion, eller benytte dig af en af de sorteringsalgoritmer der er præsenteret i undervisningen. Det er også i orden at benytte sig af F#'s `printf` format specifiers til alignment af kolonner.

```
type time = int
type state = OK of time | DEAD of time

type person = {id: int;                    (* unique id *)
               age: int;                   (* age of person *)
               state: state;               (* health state *)
               pos: vec2;                  (* position *)
               vel: vec2}                  (* velocity *)
```

Figure 2: Basale simuleringstyper.

listen, for hvilke input-strengen optræder i den enkelte fanges sidste sætning.

Opgave 2: Simulering af en pandemi

Denne delopgave omhandler simulering af en pandemi. Formålet med simuleringen er blandt andet at undersøge effekten af adfærdsændringer blandt personer.

Delopgaven tager udgangspunkt i kode der implementerer en simulation af personer der bevæger sig indenfor et to-dimensionelt bræt (et board).[4] Brættet initialiseres med et antal personer, der hver tilegnes en tilfældig position og en tilfældig alder mellem 0 og 100 år.

Positioner og hastigheder for personer modeleres med typen `vec2`:

```
type vec2 = double * double
```

Simuleringen foretages i tidsstep. En persons tilstand til tiden $t + 1$ beregnes således som en funktion af en persons tilstand til tiden t. Levende personer antages at bevæge sig med en konstant hastighed, men dog således at en person kan skifte retning tilfældigt med en vis sandsynlighed i hvert tidsstep.

Typen `person` er defineret i Figur 2 sammen med en definition af typen `time` og en definition af typen `state`. Typen `time` er defineret som et heltal, der indikerer et tidspunkt i simulationen eller en tidsperiode. Typen `state` definerer en persons helbredstilstand (`OK` i eller `DEAD` i). Bemærk at typen `state` giver mulighed for at der holdes styr på i hvor lang tid en person har været i en givet tilstand.

En konfiguration (type `conf`) er defineret som størrelsen på et bræt samt en liste af personer:

```
type board = int*int
type conf = {board:board; persons:person list}
```

[4]Koden er tilgængelig som filen `pandemic0.fs` og koden oversættes med kommandoen `fsharpc pandemic0.fs` og den resulterende eksekverbare fil `pandemic0.exe` kan køres med kommandoen `mono pandemic0.exe`.

Koden der er udleveret er centreret omkring en funktion `run`, der indeholder et `loop`, som iterativt transformerer en initiel konfiguration til en endelig konfiguration. Til hvert tidsstep opsamles der statistik (type `stat`), som kan udskrives efter en endt simulation og derved give et overblik over hvor mange personer der er døde til et givet tidspunkt. Ligeledes til hvert tidsstep beregnes der en ny konfiguration. Hver levende persons position ændres baseret på personens hastighed og hvis en person er ældre end 90 år er der en chance for at personen dør:

```
(* Function that may turn an old age OK person into a DEAD person *)
let next_state (S:settings) (C:conf) (p:person) : state =
    match p.state with
      | OK t → if p.age > 90 && rand() < S.oldAgeDeathRate then DEAD 0
               else OK (t+1)
      | DEAD t → DEAD (t+1)
```

Argumentet `S` indeholder nogle centrale settings og benyttes her til at tilgå sandsynligheden for at en person ældre end 90 år dør i det enkelte tidsstep.

Et eksempel på resultatet af en kørsel af det oversatte program `pandemic0.fs` er givet i Figur 3. Bemærk at den endelige konfiguration indeholder tre `x`'er.

De følgende spørgsmål lægger op til at udvide `pandemic0.fs` til at simulere at personer kan blive syge hvis de er i nærheden af andre syge personer og at personer der er syge efter en endt periode enten dør eller bliver raske og immune overfor sygdommen.

Spørgsmål 2.A (10 %)

Udvid typen `state` med to yderligere tilstande `ILL` og `IMMUNE`. De to yderligere helbredstilstande skal gøre det muligt at afgøre i hvormange tidssteps en person har være i den pågældende tilstand.

Som konsekvens af udvidelsen skal du udvide funktionen `pr_person` til at returnere strengen `"x"` for en død person og strengen `"$"` for en immun person.

Implementer endvidere to funktioner `isIll` samt `isImmune`, som i lighed med `isDead` kan benyttes til at undersøge om en person er i den pågældende tilstand.

Endelig skal du tilføje en 45årig syg person til den initielle konfiguration. Personen skal have `id=n`, være positioneret tilfældigt på brættet og have hastigheden `(1.0,0.0)`.

Bemærk at med de ønskede ændringer skulle du gerne kunne køre programmet og se at den endelige konfiguration (efter 20 iterationer) indeholder en syg person.[5]

Spørgsmål 2.B (10 %)

Udvid funktionen `next_state` ved at implementere følgende transitionsmuligheder:

[5]Bemærk at den simple udskrivningsrutine `pr_conf` kun viser en person i hvert felt på brættet, men at der muligvis kan være flere personer i hvert felt; det vil således være muligt at en syg person "gemmer sig under" en anden person på brættet.

```
bash-3.2$ mono pandemic0.exe
Run with T (time steps) = 20
Final Configuration:
-----------------------------------------
|   o o                                o  |
| o                             o  oo      |
|                                 o       |
|            o                     o   o   |
|              o   o               o       o|
|          o                     o        o |
|                o  o                      |
|                     ooo          o o      |
|    o   o     o                   xo    o |
|         oo                   o           |
|           o   o              o o    o    |
|     o         o   o               o      |
|o    o o       o                  o  o    |
|          o       xo         oo oo o      |
|o        o             o       o          |
|    o o   o       o    o o       o        |
|    o  o oo                 oo   o o oo   |
|               o   o  o          o   o o  |
|               oo   o        o    o     o |
|    o   ox             o                  |
-----------------------------------------
| o: ok, x: dead
-----------------------------------------

Evaluation statistics:
0:  x
1:  x
2:  x
3:  x
4:  x
5:  x
6:   x
7:   x
8:   x
9:   x
10:  x
11:  x
12:  x
13:  x
14:  x
15:   x
16:   x
17:   x
18:    x
19:    x
20:    x

Deaths: 3 / 100
```

Figure 3: Kørsel af 20 tidsstep.

1. Efter at en person har været syg i et antal sygedage (10 tidsstep) kan personens tilstand enten ændre sig til at være immun (med 90 procent sandsynlighed) eller til at være død (med 10 procent sandsynlighed).

2. En rask person kan blive syg hvis personen er indenfor 2.3 enheder af en allerede syg person (gør brug af funktionen `dist`). Smittesandsynligheden er 50 procent for hver syg person der opfylder nærhedskriteriet.

Den mest hensigtsmæssige løsning vil basere sig på at udvide record-typen `settings` med felter for antal sygedage (`illDays`), dødsraten (`deathRate`; hvor stor en rate af de syge personer dør efter `illDays`), infektionsraten (`infectionRate`; hvor stor er sandsynligheden for at en syg person tæt på giver smitte) og infektionsafstanden (`infectionDist`; hvad er den maksimale afstand der giver anledning til at smitte er mulig).

Udvid også programmet til efter endt simulering at udskrive antallet af syge personer samt antallet af immune personer i den endelige konfiguration.

Spørgsmål 2.C (10 %)

Udvid statistikvisningen (funktionen `pr_stats`) til udover at afbilde antal døde personer nu også viser antal syge personer. Du kan passende benytte dig af `padLeft` funktionen til at vise et `"i"` til venstre eller til højre for `"x"` for hvert tidsstep.

Ved at ændre de 20 tidssteps til 60 skulle du gerne se at antallet af syge personer vokser for derefter at falde igen efter tilstrækkeligt mange af de syge personer er enten døde eller blevet immune.

Spørgsmål 2.D (20 %)

For at modellere at et sygehusvæsen vil have svært ved at håndtere mange samtidige syge skal du nu implementere muligheden for at sandsynligheden for at en person dør afhænger af hvormange personer der er syge.

En løsning kan være at implementere `deathRate` som en funktion af type `int→double`, der tager det samlede antal syge personer som argument. Funktionen `next_state`, som baserer sig på dødsraten blandt syge kan så udregne det totale antal syge personer og anvende funktionen `deathRate` for at finde den konkrete dødsrate.

I de næste kørsler af programmet skal du benytte dig af en funktion for dødsraten der returnerer 0.3 såfremt antallet af syge er større end 40 og 0.1 hvis antallet af syge er mindre end eller lig med 40.

Sammenlign nu to kørsler af programmet hvor infektionsafstanden er henholdsvis 2.3 og 1.8 og forklar og diskutér dine observationer.