BSWU 1409004U **IT University, E2021**

Skriftlig eksamen, Programmer som Data
Januar 2022

Version 1.00 af January 1, 2022

Dette eksamenssæt har 10 sider. Tjek med det samme at du har alle siderne.
Eksamenssættet udleveres elektronisk fra kursets hjemmeside mandag den 3. januar 2022 kl 08:00.
Besvarelsen skal afleveres elektronisk i LearnIt senest **tirsdag den 4. januar 2022 kl 14:00** som følger:

- Besvarelsen skal uploades på kursets hjemmeside i LearnIt under **Ordinary exam assignment**.
- Der kan uploades en fil, som skal have en af følgende typer: `.txt`, `.pdf` eller `.doc`. Hvis du for eksempel laver besvarelsen i LaTeX, så generer en pdf-fil. Hvis du laver en tegning i hånden, så scan den og inkluder det skannede billede i det dokument du afleverer.

Der er 4 opgaver. For at få fulde point skal du besvare alle opgaverne tilfredsstillende.

Hvis der er uklarheder, inkonsistenser eller tilsyneladende fejl i denne opgavetekst, så skal du i din besvarelse beskrive disse og beskrive hvilken tolkning af opgaveteksten du har anvendt ved besvarelsen. Hvis du mener det er nødvendigt at kontakte opgavestiller, så send en email til `sap@itu.dk` med forklaring og angivelse af problem i opgaveteksten.

Din besvarelse skal laves af dig og kun dig, og det gælder både programkode, lexer- og parserspecifikationer, eksempler, osv., og den forklarende tekst der besvarer opgavespørgsmålene. Det er altså ikke tilladt at lave gruppearbejde om eksamen.

Din besvarelse skal indeholde følgende erklæring:

Jeg erklærer hermed at jeg selv har lavet hele denne eksamensbesvarelse uden hjælp fra andre.

Du må bruge alle bøger, forelæsningsnoter, forelæsningsplancher, opgavesæt, dine egne opgavebesvarelser, internetressourcer, lommeregnere, computere, og så videre.

Du må **naturligvis ikke plagiere** fra andre kilder i din besvarelse, altså forsøge at tage kredit for arbejde, som ikke er dit eget. Din besvarelse må ikke indeholde tekst, programkode, figurer, tabeller eller lignende som er skabt af andre end dig selv, med mindre der er fyldestgørende kildeangivelse, dvs. at du beskriver oprindelsen af den pågældende tekst (eller lignende) på en komplet og retvisende måde. Det gælder også hvis den inkluderede kopi ikke er identisk, men tilpasset fra tekst eller programkode fra lærebøger eller fra andre kilder.

Hvis en opgave kræver, at du definerer en bestemt funktion, så må du gerne **definere alle de hjælpefunktioner du vil**, men du skal definere funktionen så den har den ønskede type og giver det ønskede resultat.

Udformning af besvarelsen

Besvarelsen skal bestå af forklarende tekst (på dansk eller engelsk) der besvarer spørgsmålene, med væsentlige programfragmenter indsat i den forklarende tekst, eller vedlagt i bilag (der klart angiver hvilke kodestumper der hører til hvilke opgaver).

Vær omhyggelig med at programfragmenterne beholder det korrekte layout når de indsættes i den løbende tekst, for F#-kode er som bekendt layoutsensitiv.

Snydtjek

Til denne eksamen anvendes *Snydtjek*. Cirka 20% vil blive udtrukket af studieadministrationen i løbet af eksamen. Navne bliver offentliggjort på kursets hjemmeside tirsdag den 4. januar klokken 14:00. Disse personer skal møde op i det Zoom møde, som også opslås på kursets hjemmeside, sammen med de udtrukne navne. Man vil blive trukket ind til mødet en af gangen.

Til Snydtjek er processen, at hver enkelt kommer ind i 5 minutter, hvor der stilles nogle korte spørgsmål omkring den netop afleverede besvarelse. Formålet er udelukkende at sikre at den afleverede løsning er udfærdiget af den person, som har uploaded løsningen. Du skal huske dit studiekort.

Det er obligatorisk at møde op til snydtjek i tilfælde af at du er udtrukket. Udeblivelse medfører at eksamensbesvarelsen ikke er gyldig og kurset ikke bestået. Er man ikke udtrukket skal man ikke møde op.

Opgave 1 (25%) Micro–ML: Sets

Kapitel 5 i *Programming Language Concepts* (PLC) introducerer evaluering af et højereordens funktionssprog og kapitel 6 introducerer polymorf typeinferens. Koden der anvendes her findes i kataloget `Lectures/Lec04/Fun` i kursets git repository.

Opgaven er at udvide funktionssproget med mængder (eng. *sets*), således at vi kan evaluere udtryk der manipulerer med mængder, se eksempel `ex01` nedenfor:

```
let s1 = {2, 3} in
  let s2 = {1, 4} in
    s1 ++ s2 = {2,4,3,1}
  end
end
```

Vi har tilføjet to syntaktiske konstruktioner:

- En ikke tom mængde: $\{e_1, \ldots, e_n\}$, $n \geq 1$.
- En operator e_1 ++ e_2 som laver union på to mænger repræsenteret ved udtrykkene e_1 og e_2. Operator ++ har samme præcedens og associativitet som operatoren +.

Derudover kan vi sammenligne to mænger med den eksiserende oprator =.

1. Du skal udvide lexer `FunLex.fsl` og parser `FunPar.fsy` med support for mængder og ++ operatoren defineret ovenfor. Den abstrakte syntaks i `Absyn.fs` er udvidet med `Set` der repræsenterer et mængdeudtryk.

   ```
   type expr =
     | CstI of int
     | CstB of bool
     | Set of expr list (* Exam *)
     ...
   ```

   Du ser den abstrakte syntaks for ovenstående eksempel `ex01` nedenfor:

   ```
   > fromString @"let s1 = {2, 3} in
   -   let s2 = {1, 4} in
   -     s1 ++ s2 = {2,4,3,1}
   -   end
   - end";;
   val it : Absyn.expr =
     Let
       ("s1",Set [CstI 2; CstI 3],
        Let
          ("s2",Set [CstI 1; CstI 4],
           Prim
             ("=",Prim ("++",Var "s1",Var "s2"),
              Set [CstI 2; CstI 4; CstI 3; CstI 1])))
   >
   ```

   Vis dine rettelser og at din løsning giver tilsvarende resultat for `ex01`. Vis ydermere at du ikke kan parse nedenstående eksempel med en tom mængde:

   ```
   > fromString "let s = {} in s end";;
   System.Exception: parse error near line 1, column 10
   ...
   ```

2. Figur 4.3 på side 65 i PLC viser evalueringsregler for funktionssproget, som vi har udvidet med mængder. Nedenfor ses evalueringsregler for de nye konstruktioner:

$$(set)\ \frac{\rho \vdash e_i \Rightarrow v_i,\ 1 \leq i \leq n}{\rho \vdash \{e_1, \ldots, e_n\} \Rightarrow \texttt{SetV}\{v_1, \ldots, v_n\}} \qquad (++)\ \frac{\rho \vdash e_1 \Rightarrow \texttt{SetV}\, s_1 \quad \rho \vdash e_2 \Rightarrow \texttt{SetV}\, s_2}{\rho \vdash e_1 ++ e_2 \Rightarrow \texttt{SetV}(s_1 \cup s_2)}$$

$$(=)\frac{\rho \vdash e_1 \Rightarrow v_1 \quad \rho \vdash e_2 \Rightarrow v_2 \quad b = (v_1 = v_2)}{\rho \vdash e_1{=}e_2 \Rightarrow b}$$

Reglen *set* repræsenterer en mængde som `SetV`$\{v_1, \ldots, v_n\}$, hvor vi antager, at vi har en foreningsmængde operator $\cup$, som vi kan anvende i reglen for ++. Derudover antager vi, at vi kan sammenligne to mængder i reglen for =.

Angiv et evalueringstræ for udtrykket

```
let s = {1, 2} in s ++ {3} end
```

Du finder to eksempler på evalueringstræer i figur 4.4 og 4.5 på side 66 i PLC.

3. Udvid typen `value` og funktionen `eval` i `HigherFun.fs`, således at udtryk med mængder kan evalueres som defineret af reglerne ovenfor. Vi repræsenterer mængder med den indbyggede `Set`–type i F#, som både understøtter foreningsmængde, `Set.union`, og lighed =.

```
type value =
  | Int of int
  | Closure of string * string * expr * value env (* (f, x, fBody, fDeclEnv) *)
  | SetV of Set<value> (* Exam *)
```

Det er vigtigt, at du beskriver din implementation. Vis at din implementation virker med eksemplet `ex01` ovenfor.

4. Figur 6.1 på side 102 i PLC viser typeregler for Micro–ML. Vi antager, at vi har en type t `set` der repræsenterer en mængde med elementer af type t. Nedenfor er typereglen for *set* lavet. Reglen for *set* kræver, at alle elementer e_i i mængden har samme type t. Giv et forslag til typereglerne for ++ og =.

$$(set)\frac{\rho \vdash e_i : t,\ 1 \le i \le n}{\rho \vdash \{e_1, \ldots, e_n\} : t\ \texttt{set}} \qquad (++)\frac{\ldots}{\rho \vdash e_1 ++ e_2 : \ldots}$$

$$(=)\frac{\ldots}{\rho \vdash e_1{=}e_2 : \ldots}$$

Du skal forklare dine overvejelser, der ligger til grund for, hvordan du har defineret reglerne.

Opgave 2 (30%) Micro–C: Print Stack

Kapitel 8 i *Programming Language Concepts* (PLC) introducerer sproget micro–C. Derudover introduceres en micro–C bytekode maskine i Java (`Machine.java`). Koden der anvendes her findes i kataloget `Lectures/Lec05/MicroC` i kursets git repository.

I denne opgave tilføjer vi et nyt statement `printStack` e, som udskriver stakken på skærmen på det givne sted efter nedenstående skabelon:

```
-Print Stack <e>----------------
Stack Frame
  Lokale variable og Temporære værdier
  Base peger
  Retur adresse
Stack Frame
  ...
Global
  Globale variable
```

Hvert kald til `printStack` vil printe `-Print Stack <e>-----------`, hvor **<*e*>** er den værdi (tal), som udtrykket e evalueres til. Derefter følger aktiveringsposter (eng. *"stack frame"*) og globale variable. Stakken vokser opad. Navne på variable udskrives ikke, da de ikke er kendt af bytekode maskinen `Machine.java`. Bytekode maskinen kender heller ikke forskel på lokale variable og temporære værdier. Layout af stakken er beskrevet på figur 8.2 og 8.3 i PLC.

Betragt nedenstående program `fac.c`.

```
int nFac;
int resFac;
void main(int n) {
  int i;
  i = 0;
  nFac=0;
  while (i < n) {
    resFac = fac(i);
    i = i + 1;
  }
  printStack 42;
}

int fac(int n) {
  nFac = nFac + 1;
  printStack nFac;
  if (n == 0)
    return 1;
  else
    return n * fac(n-1);
}
```

Målet med opgaven er at få uddata svarende til nedenstående, når programmet `fac.c` afvikles med Java bytekode maskinen.

```
MicroC % java Machine fac.out 1
-Print Stack 1----------------
Stack Frame
 s[9]:  Local/Temp = 0
 s[8]:  bp = 4
 s[7]:  ret = 39
Stack Frame
 s[6]:  Local/Temp = 1
 s[5]:  Local/Temp = 0
 s[4]:  Local/Temp = 1
```

```
 s[3]:  bp = -999
 s[2]:  ret = 8
Global
 s[1]: 0
 s[0]: 1
-Print Stack 42----------------
Stack Frame
 s[5]:  Local/Temp = 1
 s[4]:  Local/Temp = 1
 s[3]:  bp = -999
 s[2]:  ret = 8
Global
 s[1]: 1
 s[0]: 1

Ran 0.028 seconds
MicroC %
```

Formatteringen defineres således

- Lokale variable og temporære værdier: " `s[`*`addr`*`]:  Local/Temp =` *`s[addr]`*"
- Basepeger: " `s[`*`addr`*`]:  bp =` *`s[addr]`*"
- Returadresse: " `s[`*`addr`*`]:  ret =` *`s[addr]`*"
- Global variabel: " `s[`*`addr`*`]:` *`s[addr]`*"

I ovenstående er `s[`*`addr`*`]` adressen og ***s[addr]*** indholdet af den pågældende staksplads der udskrives.

1. Du skal udvide lexer `CLex.fsl`, parser `CPar.fsy` og `Absyn.fs` med support for `printStack` *`e`*`;`. Du ser den abstrakte syntaks for ovenstående eksempel `fac.c` nedenfor:

```
> fromFile "fac.c";;
val it : Absyn.program =
  Prog [Vardec (TypI,"nFac");
        Vardec (TypI,"resFac");
        Fundec (None,"main",[(TypI, "n")],
                Block [Dec (TypI,"i");
                       Stmt (Expr (Assign (AccVar "i",CstI 0)));
                       Stmt (Expr (Assign (AccVar "nFac",CstI 0)));
                       Stmt (While (Prim2 ("<",Access (AccVar "i"),Access (AccVar "n")),
                                    Block [Stmt (Expr (Assign (AccVar "resFac",
                                                               Call ("fac",[Access (AccVar "i")])))));
                                           Stmt (Expr (Assign (AccVar "i",
                                                               Prim2 ("+",Access (AccVar "i"),
                                                               CstI 1))))]));
                       Stmt (PrintStack (CstI 42))]);
        Fundec (Some TypI,"fac",[(TypI, "n")],
                Block [Stmt (Expr (Assign (AccVar "nFac",
                                           Prim2 ("+",Access (AccVar "nFac"), CstI 1))));
                       Stmt (PrintStack (Access (AccVar "nFac")));
                       Stmt (If (Prim2 ("==",Access (AccVar "n"),CstI 0),
                                 Return (Some (CstI 1)),
                                 Return (Some (Prim2 ("*",Access (AccVar "n"),
                                                      Call ("fac",
                                                            [Prim2("-",Access (AccVar "n"), CstI 1)])))))))]]
>
```

   Vis dine rettelser og at din løsning giver tilsvarende resultat for `fac.c`.

2. For at udskrive stakken på køretid, implementerer vi en ny bytekode instruktion `PRINTSTACK` i `Machine.fs`:

```
type instr =
  ...
  | PRINTSTACK                          (* print stack statistics, exam    *)
```

   Vis alle rettelser til `Machine.fs` således at bytekodeinstruktionen `PRINTSTACK` kan anvendes af oversætteren i `Comp.fs`.

3. Bytekode maskinen `Machine.java` skal tilsvarende udvides med instruktionen `PRINTSTACK`:

	Instruction	Stack before		Stack after	Effect
0	`CSTI` i	s	$\Rightarrow$	s, i	Push constant i
...					
26	`PRINTSTACK`	s, v	$\Rightarrow$	s	Print stack using v in header: `-Print Stack` v`----------`

Den præcise formattering på skærmen er vist ovenfor med `fac.c` som eksempel.

Du skal implementere `PRINTSTACK` i `Machine.java`. Husk, at beskrive din implementation.

Hint: En mulig tilgang til opgaven er at løbe stakken igennem fra top til bund, hvor du benytter basepegere til at identificere hver aktiveringspost. Du ved, at en basepeger peper på første lokale variabel eller temporær værdi i aktiveringsposten. Du ved, at du har nået aktiveringsposten for funktionen `main`, når den's gamle basepeger (eng. "old base pointer") har værdien -999. På dette tidspunkt kan du finde de globale variable.

Hint: Du kan anvende nedenstående kodestruktur i `Machine.java`:

```
static int execcode(int[] p, int[] s, int[] iargs, boolean trace) {
  int bp = -999; // Base pointer, for local variable access
  ...
    case PRINTSTACK: /* Exam */
      int N = s[sp--]; /* Number to print in header line */
      printStack(N, s, bp, sp);
      break;
  ...
}

static void printStack(int N, int[] s, int bp, int sp) {
  // Go through the stack, top to bottom using base pointer
  // and old base pointers to locate stack frames.
  ...
}
```

4. Du kan anvende nedenstående oversætterskema til at implementere `PrintStack` e i `Comp.fs`:

S⟦`printStack` e⟧ =
E⟦e⟧
`PRINTSTACK`

Vis, at din implementation fungerer ved at inkludere den genererede bytekode og vise uddata ved kørsel af programmet `fac.c`. Det er vigtigt, at du beskriver din implementation.

5. Bytekoden, som du får i ovenstående opgave, vil ligne nedenstående. Koden nedenfor er struktureret således at den kan holdes op imod kildeteksten i `fac.c`. Din opgave er, udfor hver linie, `//`, at beskrive hvilke fragmenter af `fac.c` de vedrører. De første 8 linier er udfyldt som eksempel.

Hint: Du gjorde noget tilsvarende i opgave 8.1 i PLC.

```
INCSP 1;                          // nFac som global variabel
INCSP 1;                          // resFac som global variabel
LDARGS;                           // Loade parameter n fra kommandolinie
CALL (1,"L1");                    // Kalde main med n som argument.
STOP;                             // Stop ved retur fra main.
Label "L1";                                       // Main
  INCSP 1;                                        // i som lokal variabel
  GETBP; CSTI 1; ADD; CSTI 0; STI; INCSP -1;      // i = 0
  CSTI 0; CSTI 0; STI; INCSP -1;                  //
```

```
  GOTO "L4";                                    //
Label "L3";                         //
  CSTI 1;                           //
  GETBP; CSTI 1; ADD; LDI;          //
  CALL (1,"L2"); STI; INCSP -1;     //
  GETBP; CSTI 1; ADD;               //
  GETBP; CSTI 1; ADD; LDI;          //
  CSTI 1; ADD; STI; INCSP -1;       //
  INCSP 0;                          //
Label "L4";                   //
  GETBP; CSTI 1; ADD; LDI;    //
  GETBP; CSTI 0; ADD; LDI;    //
  LT; IFNZRO "L3";            //
  CSTI 42; PRINTSTACK;        //
  INCSP -1;                   //
  RET 0;                      //
Label "L2";                         //
   CSTI 0; CSTI 0; LDI; CSTI 1;     //
   ADD; STI; INCSP -1;              //
   CSTI 0; LDI; PRINTSTACK;         //
   GETBP; CSTI 0; ADD; LDI;         //
   CSTI 0; EQ; IFZERO "L5";         //
   CSTI 1; RET 1; GOTO "L6";        //
Label "L5";                                 //
  GETBP; CSTI 0; ADD; LDI;                  //
  GETBP; CSTI 0; ADD; LDI; CSTI 1; SUB;     //
  CALL (1,"L2");                            //
  MUL; RET 1;                               //
Label "L6";      //
  INCSP 0; RET 0 //
```

6. Nedenstående er en kopi af uddata for kørsel af `fac.c`, hvor der er tilføjet nogle spørgsmål, `// spørgsmål`. Du skal besvare spørgsmålene. De første to er udfyldt som eksempel.

```
MicroC % java Machine fac.out 1
-Print Stack 1----------------
Stack Frame // Funktion fac
 s[9]:  Local/Temp = 0 // Stakplads til lokal variabel n
 s[8]:  bp = 4
 s[7]:  ret = 39
Stack Frame  // Udfyld navn på funktion
 s[6]:  Local/Temp = 1 // Udfyld navn på variabel eller forklar temporær værdi.
 s[5]:  Local/Temp = 0 // Udfyld navn på variabel eller forklar temporær værdi.
 s[4]:  Local/Temp = 1 // Udfyld navn på variabel eller forklar temporær værdi.
 s[3]:  bp = -999
 s[2]:  ret = 8
Global
 s[1]: 0  // Udfyld navn på global variabel
 s[0]: 1  // Udfyld navn på global variabel
-Print Stack 42----------------
Stack Frame // Main
 s[5]:  Local/Temp = 1 // Udfyld navn på variabel eller forklar temporær værdi.
 s[4]:  Local/Temp = 1 // Udfyld navn på variabel eller forklar temporær værdi.
 s[3]:  bp = -999
 s[2]:  ret = 8
Global
 s[1]: 1  // Udfyld navn på global variabel
 s[0]: 1  // Udfyld navn på global variabel

Ran 0.028 seconds
MicroC %
```

Opgave 3 (25%) Micro–C: Intervalcheck

Kapitel 8 i *Programming Language Concepts* (PLC) introducerer sproget micro–C. Derudover introduceres en micro–C bytekode maskine i Java (`Machine.java`). Koden der anvendes her findes i kataloget `Lectures/Lec05/MicroC` i kursets git repository.

Opgaven er at udvide micro–C med *intervalcheck*, som er udtryk af formen e `within [`e_1`,`e_2`]` der returnerer sand (1) eller falsk (0) afhængig af om $v_1 \leq v \leq v_2$ er opfyldt, hvor v, v_1 og v_2 er resultaterne af at evaluere e, e_1 og e_2. Der er følgende semantiske krav til implementationen:

- Udtrykkene e, e_1 og e_2 skal altid evalueres præcis en gang.
- Nøgleordet `within` har samme præcedens som andre sammenligningsoperatorer, f.eks. < og >.

I den abstrakte syntaks repræsenteres et intervalcheck med `WithIn (`e, e_1, e_2`)`, hvor e, e_1 og e_2 er vilkårlige udtryk.

Betragt følgende micro–C program `within.c`, hvor du også kan se forventet uddata.

```
void main() {
  print (0 within [print 1,print 2]);  // Expected output: 1 2 0
  print (3 within [print 1,print 2]);  // Expected output: 1 2 0

  print (print 42 within [print 40,print 44]);   // Expected output: 40 44 1 1
  print ((print 42) within [print 40,print 44]); // Expected output: 42 40 44 1
}
```

Målet med opgaven er at få uddata svarende til nedenstående, når programmet `within.c` afvikles med Java bytekode maskinen.

```
MicroC % java Machine within.out
1 2 0 1 2 0 40 44 1 1 42 40 44 1
Ran 0.018 seconds
```

Bemærk, at `print (print 42 within [print 40,print 44]);` implicit parses som `print (print (42 within [print 40,print 44]));`.

1. Du skal udvide lexer `CLex.fsl`, parser `CPar.fsy` og `Absyn.fs` med support for udtrykket e `within [`e_1`,`e_2`]`. Du ser den abstrakte syntaks for ovenstående eksempel `within.c` nedenfor:

```
> fromFile "within.c";;
val it : Absyn.program =
  Prog [Fundec (None,"main",[],
    Block [Stmt (Expr (Prim1 ("printi", WithIn (CstI 0,
                                                Prim1("printi",CstI 1),
                                                Prim1 ("printi",CstI 2)))));
           Stmt (Expr (Prim1 ("printi", WithIn (CstI 3,
                                                Prim1 ("printi",CstI 1),
                                                Prim1 ("printi",CstI 2)))));
           Stmt (Expr (Prim1 ("printi", Prim1 ("printi", WithIn (CstI 42,
                                                                 Prim1("printi",CstI 40),
                                                                 Prim1("printi",CstI 44))))));
           Stmt (Expr (Prim1 ("printi", WithIn (Prim1 ("printi",CstI 42),
                                                Prim1 ("printi",CstI 40),
                                                Prim1 ("printi",CstI 44))))])]
```

   Vis dine rettelser og at din løsning giver tilsvarende resultat for `within.c`. Der er vigtigt, at du beskriver din løsning.

2. For at implementere oversættelsen af `within`–udtrykket i `Comp.fs` skal du lave et oversætterskema svarende til dem du finder i figur 8.6 i PLC. Du må kun anvende de bytekodeinstruktioner, som allerede findes i bytekodefortolkeren, se figur 8.1 i PLC.

   E⟦ e `within [`e_1`,`e_2`]` ⟧ =
   E⟦ e ⟧
   ...

Forklar hvorfor dit oversætterskema opfylder de semantiske krav givet ovenfor.

3. Du kan nu anvende dit oversætterskema ovenfor til at implementere oversættelsen af `within`–udtrykket i `Comp.fs`:

   Det er vigtigt, at du beskriver din implementation. Vis at din implementation fungerer ved at vise uddata ved kørsel af programmet `within.c`. I fald dit uddata ikke svarer helt til det forventede kan du beskrive hvad du tænker der er anderledes i din løsning.

Opgave 4 (20%) Icon

Kapitel 11 i *Programming Language Concepts* (PLC) introducerer "continuations" og "back tracking" samt sproget Icon. Koden der anvendes her findes i filen `Lectures/Lec11/Cont/Icon.fs` i kursets git repository.

I Icon kan vi f.eks. skrive `write(1 to 10)`. Ved at anvende implementationen i filen `Icon.fs` kan vi udtrykke dette i abstrakt syntaks:

```
let examEx1 = Write(FromTo(1,10))
```

og køre eksemplet, der udskriver tallet 1 på skærmen, inde fra F# fortolkeren:

```
> run examEx1;;
1 val it : value = Int 1
```

1. Skriv et Icon udtryk, som udskriver værdierne 1 2 3 4 5 6 7 8 9 10 på skærmen, fx.:

   ```
   > run ...;;
   1 2 3 4 5 6 7 8 9 10 val it : value = Int 0
   ```

   hvor `...` repræsenterer dit svar. Forklar hvorledes du får udskrevet alle 10 tal.

2. Skriv et Icon udtryk, som udskriver 10–tals tabellen, som en lang linie af tal.

   ```
   > run ...;;
   1 2 3 4 5 6 7 8 9 10 2 4 6 8 10 12 14 16 18 20 3 6 9 12 15 18 21 24 27
   30 4 8 12 16 20 24 28 32 36 40 5 10 15 20 25 30 35 40 45 50 6 12 18 24
   30 36 42 48 54 60 7 14 21 28 35 42 49 56 63 70 8 16 24 32 40 48 56 64
   72 80 9 18 27 36 45 54 63 72 81 90 10 20 30 40 50 60 70 80 90 100
   val it : value = Int 0
   ```

   hvor `...` repræsenterer dit svar. Du skal forklare hvordan din løsning fungerer.

3. Omskriv dit svar i opgaven ovenfor, således at 10–tals tabellen udskrives på 10 linier med 10 tal i hver linie.

   ```
   > run ...;;

    1 2 3 4 5 6 7 8 9 10
    2 4 6 8 10 12 14 16 18 20
    3 6 9 12 15 18 21 24 27 30
    4 8 12 16 20 24 28 32 36 40
    5 10 15 20 25 30 35 40 45 50
    6 12 18 24 30 36 42 48 54 60
    7 14 21 28 35 42 49 56 63 70
    8 16 24 32 40 48 56 64 72 80
    9 18 27 36 45 54 63 72 81 90
    10 20 30 40 50 60 70 80 90 100 val it : value = Int 0
   ```

   hvor `...` repræsenterer dit svar. Du skal forklare hvordan din løsning fungerer.

4. Udvid implementationen af Icon med en ny generator `Random(`*`min`*`,`*`max`*`,`*`num`*`)`, som genererer *num* tilfældige værdier i intervallet *min* og *max*, begge inklusive. Det antages, at $min \leq max$ og $num \geq 0$. Generatoren `Random` fejler med det samme, hvis $min > max$ eller $num \leq 0$.

   Eksempelvis giver

   ```
   > run (Every(Write(Random(1,10,3))));;
   10 1 5 val it : value = Int 0
   ```

   Da tallene vælges tilfældigt, vil du forventeligt se andre tal.

   Hint: Du kan bruge `randomNext` nedenfor til at udvælge tallene.

   ```
   let random = new System.Random();
   let randomNext(min, max) =
     random.Next(min,max+1)  // max is exclusive in Next.
   ```