BSWU BPRD **IT University, E2016**

Skriftlig eksamen, Programmer som Data
12.–13. januar 2017

Dette eksamenssæt har 7 sider. Tjek med det samme at du har alle siderne.
Eksamenssættet udleveres elektronisk fra kursets hjemmeside torsdag 12. januar 2017 kl 09:00.
Besvarelsen skal afleveres elektronisk i LearnIt senest **fredag 13. januar 2017 kl 14:00** som følger:

- Besvarelsen skal uploades på kursets hjemmeside i LearnIt under **Submit 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 5 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 den ønskede funktion så den har netop den type og giver det resultat som opgaven kræver.

Udformning af besvarelsen

Besvarelsen skal bestå af forklarende tekst (på dansk eller engelsk) der besvarer spørgsmålene, med nødvendige 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*. Det betyder at cirka 20% vil blive udtrukket af studeadministrationen i løbet af eksamen. Navne på disse personer vil blive offentliggjort på kursets hjemmeside fredag den 13. januar klokken 14:00. Disse personer skal stille i lokale 2A18 fredag den 13. januar klokken 16.30, dvs. et par timer efter deadline for aflevering i learnIt.

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.

Vi forventer hele processen kan gøres på lidt over 1 time.

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 %): Regulære udtryk og automater

Betragt dette regulære udtryk over alfabetet $\{d, \texttt{','}\}$, hvor d står for decimalt ciffer og `','` er komma:

$$d+ \ \texttt{','}? \ d*$$

Ved antagelse af, at *d* svarer til tallene fra 0 til 9 og `','` er et komma, så beskriver det regulære udtryk kommatal.

1. Giv en uformel beskrivelse af sproget (mængden af alle strenge) der beskrives af dette regulære udtryk. Giv mindst 4 eksempler på kommatal der genkendes af dette regulære udtryk og som understøtter din uformelle beskrivelse.

2. Konstruer og tegn en ikke-deterministisk endelig automat ("nondeterministic finite automaton", NFA) der svarer til det regulære udtryk. Husk at angive starttilstand og accepttilstand(e). Du skal enten bruge en systematisk konstruktion svarende til den i forelæsningen eller som i Introduction to Compiler Design (ICD), eller Basics of Compiler Design (BCD), eller forklare hvorfor den resulterende automat er korrekt.

3. Konstruer og tegn en deterministisk endelig automat ("deterministic finite automaton", DFA) der svarer til det regulære udtryk. Husk at angive starttilstand og accepttilstand(e). Du skal enten bruge en systematisk konstruktion svarende til den i forelæsningen eller som i Introduction to Compiler Design (ICD), eller Basics of Compiler Design (BCD), eller forklare hvorfor den resulterende automat er korrekt.

4. Angiv et regulært udtryk, der beskriver kommatal, med følgende egenskaber:

   - Der tillades kommatal uden komma, dvs. heltal.
   - Der tillades maksimalt et komma, og når der er et komma skal der også være mindst et tal foran og efter kommaet.
   - Den tomme streng genkendes også af det regulære udtryk.

   Der lægges vægt på at det regulære udtryk ikke umiddelbart kan skrives kortere.

Opgave 2 (20 %): Icon

Kapitel 11 i *Programming Language Concepts* (PLC) introducerer "continuations" og "back tracking" samt sproget Icon.

I Icon kan vi f.eks. skrive `write(1 to 6)`. Ved at anvende implementationen i filen `Icon.fs` fra lektion 10 kan vi udtrykke dette i abstrakt syntaks:

```
let examEx1 = Write(FromTo(1,6))
```

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

```
> run iconEx1;;
1 val it : value = Int 1
```

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

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

   hvor `...` repræsenterer dit svar. Forklar hvorledes du får udskrevet alle 6 tal.

2. Skriv et Icon udtryk, som udskriver følgende på skærmen:

   ```
   run ...;;
   33 34 43 44 53 54 63 64 val it : value = Int 0
   >
   ```

   hvor `...` repræsenterer dit svar.

3. Udvid implementationen af Icon med en ny generator `FromToBy(s,e,i)`, som genererer værdierne mellem s og e i hop af i. Det antages at $s <= e$ og $i >= 0$. `FromToBy` fejler med det samme, hvis $s > e$ eller $i < 0$.

   Eksempelvis giver

   ```
   > run (Every(Write(FromToBy(1, 10, 3))));;
   1 4 7 10 val it : value = Int 0
   ```

   Den eksisterende generator `FromTo(s,e)` kan implementeres med `FromToBy(s,e,1)`.

4. Skriv en udgave af dit svar til opgave 2 ovenfor, som anvender generatoren `FromToBy`.

5. Kan du få konstruktionen `FromToBy` til at generere det samme tal, fx 10, uendelig mange gange? Hvis, ja, så giv et eksempel.

Opgave 3 (15 %): Print i micro-ML

Kapitel 5 i *Programming Language Concepts* (PLC) introducerer evaluering af et højereordens funktionssprog. Opgaven er at udvide funktionssproget med muligheden for at printe værdier på skærmen.

Udtrykket `print` e skal evaluere e til en værdi v, som henholdsvis printes på skærmen samt returneres som resultat af udtrykket. Eksempelvis vil udtrykket `print 2` udskrive 2 på skærmen og returnere værdien 2.

1. I den abstrakte syntaks repræsenteres funktionen `print` med `Print` e, hvor e er et vilkårligt udtryk. Udvid typen `expr` i `Absyn.fs` med `Print` således at eksempelvis `Print(CstI 1)` repræsenterer udtrykket der printer konstanten 1 på skærmen og returnerer værdien 1.

2. Udvid lexer og parser, således at print er understøttet med syntaksen `print` e, hvor `print` er et nyt nøgleord, se funktionen `keyword` i filen `FunLex.fsl`.

   Vis den abstrakte syntaks for følgende eksempler

   ex1: `print 1`

   ex2: `print ((print 1) + 3)`

   ex3: `let f x = x + 1 in print f end`

   Vis yderligere 3 eksempler der indeholder både konstanter og funktioner.

3. Udvid funktionen `eval` i `HigherFun.fs`, med evaluering af `Print` e. Hvis v er værdien af at evaluere e, så er resultatet af `Print` e at udskrive v på skærmen samt returnere v.

   Hint: Du kan anvende F#'s indbyggede `printfn` funktion med format streng `"%A"` til at udskrive værdierne af type `value` i filen `HigherFun.fs`.

   Eksempelvis giver resultatet af at evaluere eksempel `ex3` ovenfor

   ```
   > run(fromString ex3);;
   Closure ("f","x",Prim ("+",Var "x",CstI 1),[])
   val it : HigherFun.value = Closure ("f","x",Prim ("+",Var "x",CstI 1),[])
   >
   ```

Opgave 4 (15 %): Pipes i micro-ML

Kapitel 5 i *Programming Language Concepts* (PLC) introducerer evaluering af et højereordens funktionssprog og kapitel 6 introducerer polymorf typeinferens.

Opgaven er at udvide funktionssproget med "pipes", >> og |>, kendt fra F#.

Hvis x er et argument og f en funktion, så er operatoren |> defineret ved x |> f = $f(x)$, dvs. argumentet x "pipes" ind i f. Målet er at vi i micro-ML f.eks. kan skrive

```
let f x = x + 1 in 2 |> f end
```

Operatoren >> svarer til funktionssammensætning, dvs. hvis f og g er to funktioner, så er f >> g den sammensatte funktion defineret ved (f >> g)(x) = $g(f(x))$. Eksempelvis vil

```
let f x = x + 1 in let g x = x + 2 in f >> g end end
```

returnere en ny funktion, som alt i alt lægger 3 til dens argument.

1. Udvid lexer og parser, således at operatorerne |> og >> er understøttet. Du kan eksempelvis introducere to nye tokens `PIPERIGHT` for |> og `COMPOSERIGHT` for >>.

   Begge operatorer er venstre associative og har en præcedens svarende til lighed (token `EQ` i filen `FunPar.fsy`). Det betyder bl.a. at +, −, ⋆ og / alle har højere præcedens end |> og >>.

   Du kan genanvende den abstrakte syntaks `Prim` til de to nye operatorer, hvor operanderne er henholdsvis "|>" og ">>".

   Parseren bør give følgende abstrakte syntaks for de to eksempler ovenfor:

```
Letfun ("f","x",Prim ("+",Var "x",CstI 1),
                Prim ("|>",CstI 2,Var "f"))

Letfun ("f","x",Prim ("+",Var "x",CstI 1),
                Letfun ("g","x",Prim ("+",Var "x",CstI 2),
                                Prim (">>",Var "f",Var "g")))
```

2. Anvend parseren på nedenstående 3 eksempler og forklar den genererede abstrakte syntaks udfra reglerne om præcedens og associativitet af |> og >>. Forklar yderligere, for hvert eksempel, hvorvidt du mener, at den genererede syntaks bør repræsentere et validt micro-ML program.

```
(a) let f x = x+1 in
      let g x = x+2 in
        2 |> f >> g
      end
    end
```

```
(b) let f x = x+1 in
      let g x = x+2 in
        2 |> (f >> g)
      end
    end
```

```
(c) let f x = x in
      let g x = x in
        2=2 |> (f >> g)
      end
    end
```

3. Figur 6.1 på side 98 i PLC viser typeinferensregler for funktionssproget vi har udvidet med |> og >>. Vi definerer følgende to typeregler for de nye operatorer:

$$(\text{|>})\ \frac{\rho \vdash e_1 : t_1 \quad \rho \vdash e_2 : t_1 \to t_2}{\rho \vdash e_1 \text{ |> } e_2 : t_2} \qquad (\text{>>})\ \frac{\rho \vdash e_1 : t_1 \to t_2 \quad \rho \vdash e_2 : t_2 \to t_3}{\rho \vdash e_1 \text{>> } e_2 : t_1 \to t_3}$$

   Angiv et typeinferenstræ for udtrykket

```
let f x = x + 1 in 3 |> f end
```

   Du finder to eksempler på typeinferenstræer i figur 4.8 og 4.9 på side 70 i PLC.

Opgave 5 (25 %): Tupler i List-C

I denne opgave udvider vi sproget List-C, som beskrevet i afsnit 10.7 i PLC, med tupler allokeret på hoben (*eng.* heap). Filerne der anvendes findes i `listc.zip` fra lektion 9. Spildopsamling (*eng.* Garbage collection) behøver ikke at fungere for at løse denne opgave.
Tupler kan have 1, 2 eller flere elementer. Figuren nedenfor angiver hvordan et tupel med N elementer, $v_1, \ldots, v_N$, repræsenteres på hoben.

	p	$p+1$	$p+2$		$p+N$	
...	*header*	v_1	v_2	...	v_N	...

Afsnit 10.7.3 og 10.7.4 i PLC beskriver indholdet af *header*, der bl.a. indeholder et tag og størrelsen af blokken som antal ord (*eng.* words). Et tupel fylder hermed $N+1$ ord på hoben. Som eksempel er tuplen med elementerne `31`, `32` og `33` repræsenteret nedenfor.

	p	$p+1$	$p+2$	$p+3$	
...	*header*	`31`	`32`	`33`	...

Opgaven er at udvide List-C således at vi kan oprette (funktion `tup`), opdatere (funktion `upd`) og læse fra (funktion `nth`) tupler. For at simplificere implementationen mest muligt genbruger vi typen `dynamic`, se eksempel nedenfor.

```
void main() {
  dynamic t1;
  t1 = tup(32,33,34);
  printTuple(t1,3); // 32 33 34
  upd(t1,0,42);
  printTuple(t1,3); // 42 33 34
  dynamic t2;
  t2 = tup(10,11,12,13,14,15);
  upd(t2,5,42);
  printTuple(t2,6); // 10 11 12 13 14 42
}

void printTuple(dynamic t, int n) {
  int i;
  i = 0;
  while (i < n) {
    print nth(t,i);
    i = i + 1;
  }
}
```

De tre funktioner er defineret således:

- `tup(`$e_1, \ldots, e_N$`)` allokerer et tupel i hoben med resultaterne af at evaluere $e_1, \ldots, e_N$.
- `upd(`t, i, e`)` opdaterer element i i tuplen t med værdien af at evaluere e. Første element har indeks 0.
- `nth(`t, i`)` returnerer værdien af elementet med indeks i i tuplen t.

Der er ikke noget check af at indekset i går ud over tuplen eller ej. Således kan der nemt laves programmer med uforudsigelig effekt. Opgaven er at implementere tupler, således at ovenstående program kan køres. Nedenfor følger en opskrift på en mulig implementation, som du kan tage udgangspunkt i:

`Absyn.fs`: En mulighed er at implementere de tre funktioner `tup`, `nth` og `upd` med et nyt primitiv `PrimN`(*opr*,*es*), som tager en streng *opr* og en liste af udtryk *es*. Bemærk at `tup` kan have et vilkårligt antal elementer. Ligeledes kræver `upd` tre argumenter, hvilket der ikke er support for med `Prim1` eller `Prim2`.

`CPar.fsy`: Tilføj tre nye tokens således at de tre funktioner `tup`, `nth` og `upd` let genkendes. Tilføj tre grammatikregler således der genereres en knude i den abstrakte syntaks med `PrimN` for hver af de tre funktioner. Den abstrakte syntaks for `t1 = tup(32,33,34);` kunne f.eks. være

```
Stmt(Expr(Assign(AccVar "t1",PrimN ("tup",[CstI 32; CstI 33; CstI 34]))));
```

`CLex.fsl`: Udvid f.eks. funktionen `keyword` til at returnere de tre nye tokens fra `CPar.fsy` når henholdsvis `tup`, `upd` og `nth` genkendes.

`Machine.fs`: Tilføj tre nye bytekode instruktioner `TUP`, `UPD` og `NTH` til at allokere og manipulere med tuplerne:

	Instruction	Stack before		Stack after	Effect
0	`CSTI` i	s	$\Rightarrow$	s, i	Push constant i
...					
32	`TUP` n	$s, v_1, \ldots, v_n$	$\Rightarrow$	s, p	Where n is number of elements and p is pointer to tuple.
33	`UPD`	s, p, i, v	$\Rightarrow$	s, p	Element at index i is updated with value v. The pointer p to the tuple is left on the stack.
34	`NTH`	s, p, i	$\Rightarrow$	s, v	The value v at index i in the tuple p is left on the stack.

I tabellen ovenfor er den eksisterende bytekode instruktion 0 `CSTI` medtaget til sammenligning.

`Comp.fs`: I funktionen `cExpr` oversættes de tre tilfælde af `PrimN` svarende til de tre grammatikregler der er oprettet i `CPar.fsy`. De tre bytekode instruktioner `TUP`, `UPD` og `NTH` anvendes. Bemærk funktionen `cExprs`, som bruges til at oversætte en liste af udtryk.

`listmachine.c`: Giv en tuple tagget 1. Du skal implementere de tre bytekode instruktioner `TUP`, `UPD` og `NTH`. Ved `TUP` skal du anvende `allocate` til at allokere tuplen i hoben samt lave kode som kopierer værdierne fra stakken til hoben. For `UPD` og `NTH` skal du huske at indekset i er tagget, dvs. anvende `Untag` for at få indekset ind i tuplen.

1. Vis (i udklip) de modifikationer du har lavet til filerne `Absyn.fs`, `CLex.fsl`, `CPar.fsy`, `Comp.fs`, `Machine.fs` og `listmachine.c` for at implementere tupler. Giv en skriftlig forklaring på modifikationerne.
2. Dokumenter ved at køre ovenstående eksempelprogram, at du får følgende uddata: `32 33 34 42 33 34 10 11 12 13 14 42`.
3. Antag at vi ønsker at implementere et primitiv `printTuple(t)`, som kan udskrive indholdet af en tupel t på skærmen. Resultatet af `printTuple(t)` er tuplen selv. Giv en opskrift, på niveau med ovenstående opskrift, der forklarer hvorledes du vil gøre dette. Det er tilladt at ændre repræsentationen af tupler på hoben. Det er *ikke* et krav, at du implementerer `printTuple`.