BSWU BPRD **IT University, E2014**

Skriftlig eksamen, Programmer som Data
12.–13. januar 2015

Dette eksamenssæt har 6 sider. Tjek med det samme at du har alle siderne.
Eksamenssættet udleveres elektronisk fra kursets hjemmeside mandag 12. januar 2015 kl 09:00.
Besvarelsen skal afleveres elektronisk i LearnIt senest **tirsdag 13. januar 2015 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 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 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 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.

Opgave 1 (25 %): Regulære udtryk og automater

Betragt dette regulære udtryk over alfabetet $\{e, f, d, x\}$:

$$e(fd*) * x$$

Ved antagelse, at

e	svarer til	*enter bar*
f	svarer til	*fetch beer*
d	svarer til	*drink beer*
x	svarer til	*exit bar*

så beskriver det regulære udtryk strenge, der symboliserer scenarier for en gæst i fredagsbaren.

1. Giv nogle eksempler på strenge der beskrives af dette regulære udtryk. Giv en uformel beskrivelse af sproget (mængden af alle strenge) der beskrives af dette regulære udtryk.

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. Betragt følgende uendelige mængde af strenge over alfabetet $\{f, s, d\}$:

```
f
 fs
  fss
   ...
  fsd
   ...
 fd
  fds
   ...
  fdd
   ...
```

   I figuren svarer strenge med indryk n til alle strenge af længde $n + 1$. For eksempel er strengene `fs` og `fd` samtlige strenge af længde 2. Angiv et regulært udtryk der beskriver denne mængde af strenge. For at blive i analogien med fredagsbaren kan du antage at *s* svarer til *spill beer*. For eksempel vil strengen `fdss` svare til *fetch beer*, *drink beer*, *spill beer*, *spill beer*.

Opgave 2 (25 %): Par-udtryk i Funktionssprog

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 par-udtryk, således at de kan evalueres (funktion `eval` i `HigherFun.fs`).

1. Udvid typen `expr` i `Absyn.fs` med tre nye konstruktører `Pair`, `Fst` og `Snd`, således at

   - par-udtryk kan genereres, for eksempel `Pair(CstI 1, CstB true)`.
   - første komponenten kan hentes ud af et par-udtryk, for eksempel `Fst(Pair(CstI 1, CstB true))`.
   - anden komponenten kan hentes ud af et par-udtryk, for eksempel `Snd(Pair(CstI 1, CstB true))`.

   Vis (i udklip) de modifikationer du har lavet til `Absyn.fs`

2. Udvid typen `value` og funktionen `eval` i `HigherFun.fs`, således at udtryk der anvender `Pair`, `Fst` og `Snd` kan evalueres. Bemærk at `Fst` og `Snd` skal fejle, hvis værdien de anvendes på ikke er et par-udtryk. Vis (i udklip) de modifikationer du har lavet til `HigherFun.fs` og giv en skriftlig forklaring af modifikationerne på 5–10 linjer.

3. Erklær mindst fire eksempler på udtryk af type `expr` hvori `Pair`, `Fst` og `Snd` indgår. Eksemplerne skal inkludere nestede par-udtryk og mindst et par-udtryk der indeholder en funktion. Vis resultatet af at evaluere eksemplerne med `eval`. Du kan benytte funktionen `run` i `ParseAndRun.fs`.

4. Udvid lexer og parser, således at par-udtryk er understøttet. Det skal for eksempel være muligt at skrive `"(1,true)"`, `"fst(1, true)"` og `"snd(1, true)"` svarende til eksemplerne ovenfor (pind 1). Funktionerne `fst` og `snd` skal fejle, hvis værdien de anvendes på ikke er et par-udtryk. Vis (i udklip) de modifikationer du har lavet til `FunLex.fsl` og `FunPar.fsy` og giv en skriftlig forklaring af modifikationerne på 5–10 linjer.

5. Omskriv dine eksempler fra ovenfor (pind 3) således at de giver samme resultat når lexer og parser anvendes inden evaluering. Vis resultatet af at evaluere eksemplerne.

6. Figur 6.1 på side 98 i PLC viser typeinferensregler for funktionssproget vi har udvidet med par-udtryk af formen (e_1, e_2). Den tilsvarende par-type er $t * u$, således at (e_1, e_2) har type $t * u$ hvis e_1 har type t og e_2 har type u. Dette kan beskrives med følgende tre typeregeler der også dækker `fst` og `snd`:

$$(\textit{pair})\frac{\rho \vdash e_1 : t \quad \rho \vdash e_2 : u}{\rho \vdash (e_1, e_2) : t * u} \qquad (\textit{fst})\frac{\rho \vdash e : t * u}{\rho \vdash \texttt{fst}(e) : t} \qquad (\textit{snd})\frac{\rho \vdash e : t * u}{\rho \vdash \texttt{snd}(e) : u}$$

   Angiv et typeinferenstræ for udtrykket `snd(32 < 2, (10 + 2,fst(false,1+3)))`. Du finder to eksempler på typeinferenstræer i figur 4.8 og 4.9 på side 70 i PLC.

Opgave 3 (25 %): Længde af Arrays i micro-C

Som forklaret til forelæsningen den 25. september (lecture06.pdf, slide 20), og som det fremgår af `Comp.fs`, så repræsenterer micro-C et *n*-element array `int arr[n]` ved hjælp af $n + 1$ stakpladser, således:

	q	$q+1$		$q+n-1$	a	
...	`arr[0]`	`arr[1]`	...	`arr[n-1]`	q	...

Her er `arr[0]` ... `arr[n-1]` arrayets elementer, der efterfølges af q, som er adressen på `arr[0]`. Du finder koden for dette i `Comf.fs`, funktion `allocate`.

Denne repræsentation kan udnyttes til, på køretid, at beregne længden af arrayet. Hvis a er adressen på stakpladsen der indeholder q, så er $(a - q)$ længden af arrayet.

Bemærk at det kun virker hvis `arr` er erklæret og allokeret som et array, enten globalt eller lokalt, i micro-C. Det duer ikke hvis `arr` er et parameteroverført array eller hvis man har en anden vilkårlig pointer til arrayet.

I opgavebesvarelsen nedenfor skal du ignorere disse begrænsninger og blot håndtere de beskrevne tilfælde, hvor det virker.

Opgaven er at udvide micro-C med et nyt primitiv `|arr|`, som beregner længden af arrayet *arr*, forudsat at *arr* opfylder betingelserne ovenfor.

Følgende program udskriver 4 og 5 når det køres:

```
int a[4];
void main() {
  int b[5];
  print |a|;
  print |b|;
}
```

For at implementere dette primitiv `|arr|` skal du udvide den abstrakte maskine med en ny ordre `ARRLEN` der beregner længden af arrayet, og du skal modificere micro-C oversætteren lidt.

Virkningen af den nye ordre kan beskrives som i tabellen side 140 i bogen *Programming Language Concepts*:

	Instruction	**Stack before**		**Stack after**	**Effect**
0	`CSTI` i	s	$\Rightarrow$	s, i	Push constant i
...					
26	`ARRLEN`	s, a	$\Rightarrow$	s, l	Where l is the size of the array a

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

I den nye ordre 26 `ARRLEN` er a adressen på stakpladsen lige efter arrayets elementer, og denne plads indeholder tallet $q = s[a]$ som er adressen på arrayets element 0, sådan at arrayets længde beregnes som $l = a - q$. Ordren `ARRLEN` skal blot lægge arrayets længde l på stakken.

For at tilføje den nye ordre til den abstrakte maskine skal du tilpasse både filen `Machine.fs` og filen `Machine.java`. Vink: Se efter hvordan en eksisterende ordre, såsom `STOP`, håndteres i disse to filer.

Du skal også ændre micro-C oversætteren i `CLex.fsl`, `CPar.fsy` og `Comp.fs`. Det er ikke nødvendigt at ændre `Absyn.fs`, da du for eksempel kan introducere `Prim1("arrlen", ...)`, og matche ud på den i `Comp.fs`.

1. Vis (i udklip) de modifikationer du har lavet til `CLex.fsl`, `CPar.fsy`, `Comp.fs`, `Machine.fs` og `Machine.java`. Giv en skriftlig forklaring af modifikationerne på 20–40 linjer.

2. Skriv et micro-C testprogram baseret på denne skitse, hvor `sum` skal indeholde summen af elementerne i `arr` og printes:

```
void main() {
  int arr[3];
  ... initialisering af arr og andre variable...
  while (... < |arr|) {
  ...
  }
  print sum;
}
```

3. Forklar hvorfor nedenstående program printer en forkert størrelse af `a`.

```
void main() {
  int a[4];
  printlen(a);
}

void printlen(int b[]) {
  print |b|;
}
```

4. Forklar hvorfor nedenstående program printer den korrekte størrelse af `a`.

```
void main() {
  int a[40];
  printlen(&a);
}

void printlen(int *b[]) {
  print |*b|;
}
```

Opgave 4 (25 %): Initialisering af Arrays i micro-C

I F# kan man generere arrays ved hjælp af *range expressions*, for eksempel.

```
> let array = [|1 .. 2 .. 20|];;

val array : int [] = [|1; 3; 5; 7; 9; 11; 13; 15; 17; 19|]
```

En *range expression* kan beskrives $[\![b..s..e]\!]$, hvor b angiver den første værdi (*base*), s er det der lægges til (*step*) og e den maksimale tilladte værdi (*end*). Værdierne b, s og e må være både positive og negative, med undtagelse af s, der ikke må være nul.

Udvid micro-C med array erklæringer af formen `int a[`b`..`s`..`e`];`, som erklærer et nyt array `a`. I micro-C er b, s og e alle konstanter af type `int`. I micro-C anvendes samme semantik for *range expressions* som i F#. Størrelsen af arrayet er defineret som antallet af elementer i F# arrayet `[|`b`..`s`..`e`|]`, herefter kaldet arr. Alle elementer i arrayet initialiseres til netop de værdier som arr har, det vil sige, `a[0]` har samme værdi som arr`.[0]`, `a[1]` har samme værdi som arr`.[1]`, osv.

Eksempelvis, vil programmet nedenfor

```
int a[1 .. 2 .. 20];
void main() {
  int i;
  i = 0;
  while (i < 10) {
    print a[i];
    i=i+1;
  }
}
```

printe "`1 3 5 7 9 11 13 15 17 19`", svarende til eksemplet med `array` i F# ovenfor.

1. Vis (i udklip) de modifikationer du har lavet til `CLex.fsl`, `CPar.fsy`, `Absyn.fs` og `Comp.fs`. Giv en skriftlig forklaring af modifikationerne på 20–40 linjer.

   Hint: Sørg for at generere værdierne til elementerne i `Comp.fs` ved at anvende F#'s indbyggede *range expressions*, for eksempel for lister: `[1..2..20]`. På den måde får micro-C samme semantik.

   Hint: Du kan benytte ordren `CSTI` til at lægge de nødvendige værdier i arrayet.

2. Angiv et eller flere testprogrammer der erklærer arrays med forskellige kombinationer af b, s og e, både positive og negative.

3. Vis resultatet fra en kørsel af testprogrammerne og forklar, på 5 – 10 linjer, i hvilken grad resultaterne er som forventet.