Foundations1 assignment 2018
Submit by Monday of week 9 (5 November 2018)
worth 15%
Sumit to the coursework submission box before 16:00 hours
Submit only typed material
No handwritten material
Submit your entire SML program by email by the deadline

Throughout the assignment, assume the terms and definitions given in the DATA SHEET.

1. Just like the translation function $\omega : \mathcal{M} \mapsto \Lambda$ is defined in the DATA SHEET, give translation functions $\omega_1 : \Lambda \mapsto \mathcal{M}$ and $\omega_2 : \mathcal{M} \mapsto \Lambda'$ and $\omega_3 : \Lambda' \mapsto \mathcal{M}'$ and $V_1 : \mathcal{M}' \mapsto \mathcal{M}$. Your translation functions need to be complete with all subfunctions and needed information (just like ω was complete with all needed information). Submit all these functions here. (2)

2. For each of the SML terms vx, vy, vz, t1, $\cdots$ t9 in `http://www.macs.hw.ac.uk/~fairouz/foundations-2018/slides/data-files.sml`, let the overlined term represent the corresponding term in $\mathcal{M}$. I.e., $\overline{\text{vx}} = x$, $\overline{\text{vy}} = y$, $\overline{\text{vz}} = z$, $\overline{\text{t1}} = \lambda x.x$, $\overline{\text{t2}} = \lambda y.x$, $\cdots$.
   For each of vx, vy, vz, t1, t2, $\cdots$ t9, give $\overline{\text{vx}}$, $\overline{\text{vy}}$, $\overline{\text{vz}}$, $\overline{\text{t1}}$, $\overline{\text{t2}}$, $\cdots \overline{\text{t9}}$ in $\mathcal{M}$, and the translation into the corresponding terms of Λ (using $\omega : \mathcal{M} \mapsto \Lambda$), of Λ' (using $\omega_2 : \mathcal{M} \mapsto \Lambda'$), of $\mathcal{M}'$ (using $\omega_3 : \Lambda' \mapsto \mathcal{M}'$), and into $\mathcal{M}$ (using $\omega_1 : \Lambda \mapsto \mathcal{M}$ and not using $V_1 : \mathcal{M}' \mapsto \mathcal{M}$).

   Your output should be tidy as follows:

	$\bar{\cdot}$	ω	ω_2	ω_3	ω_1
t_1	$\lambda x.x$	$\lambda 1$	$[\,]1$	$[x]x$	$\lambda x.x$

(1)

3. Just like SML terms vx, vy, vz, t1, t2, $\cdots$ t9 are introduced which implement terms in $\mathcal{M}$, please implement the corresponding terms each of the other sets $\mathcal{M}'$, Λ, Λ'. Your output must be like the output in `http://www.macs.hw.ac.uk/~fairouz/foundations-2018/slides/data-files.sml`, for the implementation of these terms of $\mathcal{M}$. I.e., your output for each set must be similar to the following: (1)

   The implementation of terms in $\mathcal{M}$ is as follows:

```
val vx = (ID "x");
val vy = (ID "y");
val vz = (ID "z");
val t1 = (LAM("x",vx));
val t2 = (LAM("y",vx));
val t3 = (APP(APP(t1,t2),vz));
val t4 = (APP(t1,vz));
val t5 = (APP(t3,t3));
val t6 = (LAM("x",(LAM("y",(LAM("z",
                       (APP(APP(vx,vz),(APP(vy,vz))))))))));
val t7 = (APP(APP(t6,t1),t1));
val t8 = (LAM("z", (APP(vz,(APP(t1,vz))))));
val t9 = (APP(t8,t3));
```

4. For each of $\mathcal{M}'$, Λ, Λ', implement a printing function that prints its elements nicely and you need to test it on every one of the corresponding terms vx, vy, vz, t1, t2, $\cdots$ t9. Your output for each such set must be similar to the one below (1)

```
(*Prints a term in classical lambda calculus*)
fun printLEXP (ID v) =
    print v
  | printLEXP (LAM (v,e)) =
    (print "(\\";
     print v;
     print ".";
     printLEXP e;
     print ")")
  | printLEXP (APP(e1,e2)) =
    (print "(";
     printLEXP e1;
     print " ";
     printLEXP e2;
     print ")");
```

Printing these $\mathcal{M}$ terms yields:

```
-printLEXP vx;
xval it = () : unit

-printLEXP vy;
yval it = () : unit

-printLEXP vz;
zval it = () : unit

-printLEXP t1;
(\x.x)val it = () : unit

-printLEXP t2;
(\y.x)val it = () : unit

-printLEXP t3;
(((\x.x) (\y.x)) z)val it = () : unit

-printLEXP t4;
((\x.x) z)val it = () : unit

-printLEXP t5;
((((\x.x) (\y.x)) z) (((\x.x) (\y.x)) z))val it = () : unit

-printLEXP t6;
(\x.(\y.(\z.((x z) (y z)))))val it = () : unit

-printLEXP t8;
(\z.(z ((\x.x) z)))val it = () : unit

-printLEXP t9;
((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z))val it = () : unit
```

5. Implement in SML the translation functions ω, ω_1, ω_2, ω_3 and V_1 and give these implemented functions here. (2)

6. Test these functions on the translations of all the given terms vx, vy, vz, t1, $\cdots$ t9 and give your output clearly.

For example, if we implement itran (a translation function from $\mathcal{M}$ to $\mathcal{M}'$), then printIEXP prints expressions in $\mathcal{M}'$. Hence,

```
- printIEXP (itran t5);
<<z><[y]x>[x]x><z><[y]x>[x]xval it = () : unit
```

You need to show how all your terms are translated in all these sets and how you print them. (2)

7. Translate $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ in each of $\mathcal{M}'$, Λ and Λ' and give the SML implementation of all these translations. (1)

8. Give an implementation of leftmost reduction with a counter in $\mathcal{M}$ and test it on a number of examples. For example:

```
-  countprintloreduce t5;
((((\x.x) (\y.x)) z) (((\x.x) (\y.x)) z))-->
(((\y.x) z) (((\x.x) (\y.x)) z))-->
(x (((\x.x) (\y.x)) z))-->
(x ((\y.x) z))-->
(x x)
Number of Steps: 4
val it = () : unit
```

(2)

9. Give an implementation of rightmost reduction with a step counter in $\mathcal{M}$ and test it on a number of examples similarly to what you did for the above question. (2)

10. Test your implementations of leftmost and rightmost on a number of examples to deduce which is more efficient and which is more likely to terminate. (1)

DATA SHEET

At `http://www.macs.hw.ac.uk/~fairouz/foundations-2018/slides/data-files.sml`, you find an implementation in SML of the set of terms $\mathcal{M}$ and many operations on it. You can use all of these in your assignment. Anything you use from elsewhere has to be well cited/referenced.

† The syntax of the classical λ-calculus is given by $\mathcal{M} ::= \mathcal{V} \mid (\lambda\mathcal{V}.\mathcal{M}) \mid (\mathcal{M}\mathcal{M})$.
We assume the usual notational conventions in $\mathcal{M}$ and use the reduction rule:
$\underline{(\lambda v.P)Q} \rightarrow_\beta P[v := Q]$.

† The syntax of the λ-calculus in item notation is given by $\mathcal{M}' ::= \mathcal{V} \mid [\mathcal{V}]\mathcal{M}' \mid \langle\mathcal{M}'\rangle\mathcal{M}'$.
We use the reduction rule: $\underline{\langle Q'\rangle[v]}P' \rightarrow_{\beta'} [x := Q']P'$.

† In $\mathcal{M}$, (PQ) stands for the application of function P to argument Q.
† In $\mathcal{M}'$, $\langle Q'\rangle P'$ stands for the application of function P' to argument Q' (note the reverse order).

† The syntax of the classical λ-calculus with de Bruijn indices is given by
$\Lambda ::= \mathbb{N} \mid (\lambda\Lambda) \mid (\Lambda\Lambda)$.

† For $[x_1, \cdots, x_n]$ a list (not a set) of variables, we define $\omega_{[x_1,\cdots,x_n]} : \mathcal{M} \mapsto \Lambda$ inductively by:
$\omega_{[x_1,\cdots,x_n]}(v_i) = \min\{j : v_i \equiv x_j\}$
$\omega_{[x_1,\cdots,x_n]}(AB) = \omega_{[x_1,\cdots,x_n]}(A)\omega_{[x_1,\cdots,x_n]}(B)$
$\omega_{[x_1,\cdots,x_n]}(\lambda x.A) = \lambda\omega_{[x,x_1,\cdots,x_n]}(A)$

Hence $\omega_{[x,y,x,y,z]}(x) = 1$, $\omega_{[x,y,x,y,z]}(y) = 2$ and $\omega_{[x,y,x,y,z]}(z) = 5$.
Also $\omega_{[x,y,x,y,z]}(xyz) = 1\ 2\ 5$.
Also $\omega_{[x,y,x,y,z]}(\lambda xy.xz) = \lambda\lambda 2\ 7$.

Assume our variables are ordered as follows: $v_1, v_2, v_3, \cdots$.
We define $\omega : \mathcal{M} \mapsto \Lambda$ by $\omega(A) = \omega_{[v_1,\cdots,v_n]}(A)$ where $FV(A) \subseteq \{v_1, \cdots, v_n\}$.
So for example, if our variables are ordered as $x, y, z, x', y', z', \cdots$ then $\omega(\lambda xyx'.xzx') = \omega_{[x,y,z]}(\lambda xyx'.xzx') = \lambda\omega_{[x,x,y,z]}(\lambda yx'.xzx') = \lambda\lambda\omega_{[y,x,x,y,z]}(\lambda x'.xzx') = \lambda\lambda\lambda\omega_{[x',y,x,x,y,z]}(xzx') = \lambda\lambda\lambda 3\ 6\ 1$.

† The syntax of the λ-calculus in item notation is given by
$\Lambda' ::= \mathbb{N} \mid [\,]\Lambda' \mid \langle\Lambda'\rangle\Lambda'$.

† Assume the following SML datatypes which implement $\mathcal{M}$, Λ, $\mathcal{M}'$ and Λ' respectively (here, if `e1` implements A'_1 and `e2` implements A'_2, then `IAPP(e1,e2)` implements $\langle A'_1\rangle A'_2$ which stands for the function A'_2 applied to argumentA'_1):

```
datatype LEXP =
   APP of LEXP * LEXP | LAM of string *  LEXP |  ID of string;
```

```
datatype BEXP =
   BAPP of BEXP * BEXP | BLAM of BEXP |  BID of int;

datatype IEXP =
   IAPP of IEXP * IEXP | ILAM of string *  IEXP |  IID of string;

datatype IBEXP =
   IBAPP of IBEXP * IBEXP | IBLAM of    IBEXP |  IBID of int;
```