1	Don’t write anything on this sheet. Functions listed on the back of this sheet can be freely used. The Theory(**A**) and Practice(**B**) tasks worth 7-7 points, Programming(**C**) worth 21 points.

A. Explain the following concepts and show examples:
atomic and compound data types
n-tuple, record (Show the differences and similarities and how you can access their fields.)

B. 1st and 2nd task: What is the value of `x` after evaluating the following value-definitions?
3rd task: There are exactly two semantic errors in the following syntactically correct SML expression. Which are these? (The value definions are independent, the standard libraries are already loaded.)

1. `val x = ((Math.cos Math.pi, 3), #1 (2, 2.0), chr(3 + ord #"a"))`
2. `val x = map (String.fields Char.isAlpha) ["4aa55bb666", "56", "b"]`
3. `let val (x, y, z) = (#"x", "y", 5, 4.0) in [ord x + ord y, z] end`

C. Write the auxiliary function first, then, using it, solve the whole task. Pay attention to fulfill both specifications.

Auxiliary function: Assume the following datatype definition:

```
datatype ’a tree = E | L of ’a | N of ’a tree * ’a tree * ’a tree
```

Write two functions called `existsL` and `numberE`, which work on `’a tree lists`. `existsL` returns `1` if there exists a tree of type `L` in its argument list, `0` otherwise. `numberN` returns the number of `E` elements are there in the list.

```
existsL fs = 1, if fs contains trees of type L _, 0 otherwise
dbE fs = the number of E’s in fs
existsL : ’a tree list -> int
numberE : ’a tree list -> int
```

```
Example: existsL []                    = 0
         existsL [E, L 1, N(E,E,E)]    = 1
         existsL [L 2, N(L 3,L 4,L 5)] = 1

         numberE  []                   = 0
         numberE  [E, N(E,L #"a",E)]   = 1
         numberE  [E, N(L 5,E,L 2), E] = 2
```

Whole task: Write a function called `LbrotherE` using `existsL` and `numberE`, which when applied to an `’a tree`, returns the number of those `E` leafs, which have at least one `L` brother. Brothers are those `L` and `E` trees, which ’hang’ on a common `N` node.

Don’t define further auxiliary functions!

```
LbrotherE f = the number of those E’s in f, which have at least one L brother
LbrotherE : ’a tree -> int
```

```
Example: LbrotherE E                                = 0
         LbrotherE (L 3)                            = 0
         LbrotherE (N(E,E,E))                       = 0
         LbrotherE (N(L 9, E, N(L 5,E,E)))          = 3
         LbrotherE (N(L 7, N(L 8, N(L 9,E,E), E), E)) = 4
```

Type of standard functions

Name	*Module*	*Type*	*Name*	*Module*	*Type*
::	List	’a * ’a list -> ’a list	isSpace	Char	char -> bool
@	List	’a list * ’a list -> ’a list	isUpper	Char	char -> bool
^	String	string -> string -> string	last	List	’a list -> ’a
all	List	(’a -> bool) -> ’a list -> bool	length	List	’a list -> int
app	List	(’a -> unit) -> ’a list -> unit	ln	Math	real -> real
before	General	’a * ’b -> ’a	map	List	(’a -> ’b) -> ’a list -> ’b list
ceil	General	real -> int	mapPartial	List	(’a -> ’b option) -> ’a list -> ’b list
chr	Char	int -> char	mod	Int	int * int -> int
compare	Char	char * char -> order	nth	List	’a list * int -> ’a
compare	Int	int * int -> order	o	General	(’a -> ’b) * (’c -> ’a) -> ’c -> ’b
compare	Real	real * real -> order	ord	Char	char -> int
compare	String	string * string -> order	partition	List	(’a -> bool) -> ’a list -> ’a list * ’a list
concat	List	’a list list -> ’a list	pi	Math	real
cos	Math	real -> real	pow	Math	real * real -> real
div	Int	int * int -> int	print	TextIO	string -> unit
drop	List	’a list * int -> ’a list	printVal	Meta	’a -> ’a
exists	List	(’a -> bool) -> ’a list -> bool	quot	Int	int * int -> int
exp	Math	real -> real	real	General	int -> real
explode	String	string -> char list	rem	Int	int * int -> int
fields	String	(char -> bool) -> string -> string list	rev	List	’a list -> ’a list
filter	List	(’a -> bool) -> ’a list -> ’a list	revAppend	List	’a list * ’a list -> ’a list
find	List	(’a -> bool) -> ’a list -> ’a option	round	General	real -> int
floor	General	real -> int	sin	Math	real -> real
foldl	List	(’a * ’b -> ’b) -> ’b -> ’a list -> ’b	size	String	string -> int
foldr	List	(’a * ’b -> ’b) -> ’b -> ’a list -> ’b	sqrt	Math	real -> real
fromString	Int	string -> int option	str	String	char -> string
fromString	Real	string -> real option	sub	String	string * int -> char
hd	List	’a list -> ’a	take	List	’a list * int -> ’a list
implode	String	char list -> string	tl	List	’a list -> ’a list
isAlpha	Char	char -> bool	toLower	Char	char -> char
isAlphaNum	Char	char -> bool	toString	Int	int -> string
isDigit	Char	char -> bool	toString	Real	real -> string
isHexDigit	Char	char -> bool	toUpper	Char	char -> char
isLower	Char	char -> bool	tokens	String	(char -> bool) -> string -> string list
isPrefix	String	string -> string -> bool	trunc	General	real -> int
isPunct	Char	char -> bool	valOf	Option	’a option -> ’a
isSome	Option	’a option -> bool			

ASCII codes of characters (frotm `chr 32` to `chr 126`)

32 - 63	␣ ! " # $ % & ’ ( ) * + , − . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
64 - 95	@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _
96 - 126	` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~