1.下列模式能否与类型为 int list 的 L 匹配成功？如果匹配不成功，指出该模式的类型？（假设 x 为 int 类型）

x::L　　非空 list

成功，标准的非空列表模式

_::_　　非空 list

成功，同样是非空列表模式

x::(y::L)

成功，匹配至少含两个元素的列表

(x::y)::L

不成功，x::y 匹配第一个元素，其本身是一个列表，而不是 int，也就是说它是 int list list

[x, y]

不一定成功，其只能匹配恰包含两个元素的列表、

2. 试写出与下列表述相对应的模式。如果没有模式与其对应，试说明原因。

list of length 3

[x, y, z]

lists of length 2 or 3

没有对应模式，SML 中的一个模式只能描述一种结构，不能同时描述两种或多种不同的结构

Non-empty lists of pairs

(x, y) :: L

Pairs with both components being non-empty lists

(x::xs, y::ys)

3. 分析下述程序段（左边括号内为标注的行号）：

```
(1)     val x : int = 3
(2)     val temp : int = x + 1
(3)     fun assemble (x : int, y : real) : int =
(4)         let val g : real = let val x : int = 2
(5)                            val m : real = 6.2 * (real x)
(6)                            val x : int = 9001
(7)                            val y : real = m * y
(8)                        in y  -  m
(9)                        end
(10)        in
(11)          x + (trunc g)
```

(12)　　end
(13)
(14)　　val z = assemble (x, 3.0)

试问：第 4 行中的 x、第 5 行中的 m 和第 6 行中的 x 的声明绑定的类型和值分别为什么？第 14 行表达式 assemble(x, 3.0)计算的结果是什么？

第 4 行的 x: 类型为 int，值为 2。它是一个新的局部变量，作用域仅限于内层的 let 块。

第 5 行的 m: 类型为 real，值为 12.4。它通过 6.2 乘以第 4 行的 x 计算得出。

第 6 行的 x: 类型为 int，值为 9001。这个变量遮蔽了第 4 行的 x

assemble(x, 3.0)=27,外部 x=3，等价于调用 assemble(3,3.0),函数的局部作用域中，y=m*y=12.4*3,g=y-m=12.4*3-12.4=24.8,函数返回值 x+(trunc g)=3+24=27

4. 编写函数实现下列功能：

（1）zip: string list * int list -> (string * int) list

其功能是提取第一个 string list 中的第 i 个元素和第二个 int list 中的第 i 个元素组成结果 list 中的第 i 个二元组。如果两个 list 的长度不同，则结果的长度为两个参数 list 长度的最小值。

（2）unzip: (string * int) list -> string list * int list

其功能是执行 zip 函数的反向操作，将二元组 list 中的元素分解成两个 list，第一个 list 中的元素为参数中二元组的第一个元素的 list，第二个 list 中的元素为参数中二元组的第二个元素的 list。

对所有元素 L1: string list 和 L2: int list，unzip(zip (L1, L2)) = (L1, L2)是否成立？如果成立，试证明之；否则说明原因。

```
(*** Begin ***)
 (* zip : string list * int list -> (string * int) list *)
 fun zip ([], _) = []
   | zip (_, []) = []
   | zip (s::ss, i::ii) = (s, i) :: zip(ss, ii);

 (* unzip : (string * int) list -> string list * int list *)
  fun unzip ([]) = ([], [])
    | unzip ((s, i)::xs) =
        let
            val (ss, ii) = unzip(xs)
        in
            (s::ss, i::ii)
        end;
(*** End ***)
```

只有当 L1 和 L2 长度相等时成立，当长度不等时，较长的部分会被截断，当 unzip 时，也只能得到剩下的一部分，之前截断的超出部分会被丢弃

5. 指出下列代码的错误：

```
(* pi: real *)
val pi : real = 3.14159;

(* fact: int -> int *)
fun fact (0 : int) : int = 1
   | fact n = n * (fact (n - 1));

(* f : int -> int *)
fun f (3 : int) : int = 9
      f _ = 4;缺少管道符|

(* circ : real -> real *)
fun circ (r : real) : real = 2 * pi * r   不同类型不能直接相乘，应为 2.0

(* semicirc : real -> real *)
fun semicirc : real = pie * r     pie 未定义，应为 pi

(* area : real -> real *)
fun area (r : int) : real = pi * r * r   r 类型应为 real 而不是 int
```

6. 分析下面菲波拉契函数的执行性能

```
fun fib n = if n<=2 then 1 else fib(n-1) + fib(n-2);
```

朴素递归，每次产生两个子调用，大量数据被重复计算，时间复杂度为指数级

```
fun fibber (0: int) : int * int = (1, 1)
   | fibber (n: int) : int * int =
         let val (x: int, y: int) = fibber (n-1)
         in (y, x + y)
         End
```

尾递归，避免了冗余计算，通过元组（x,y)存储一对斐波拉契数，每次递归更新元组，时间复杂度 O(n)

借助：对所有非负整数 k，

$fib(2k) = fib(k)(2fib(k + 1) − fib(k))$

$fib(2k + 1) = fib(k + 1)2 + fib(k)2$

利用以上数学性质和分治思想，递归计算 F(n/2)和 F(n/2+1),每次递归问题规模减半，时间复杂度为 O(log n)，达到对数级别