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
_::_	非空list
x::(y::L)	
(x::y)::L	
[x, y]	

解答：逐句分析如下：

（1）x :: L，可以和 int list 匹配成功，这句话表示非空 list。

（2）_ :: _，可以和 int list 匹配成功，这句话表示非空 list。

（3）x :: (y :: L)，可以和 int list 匹配成功，这句话表示元素个数大于等于 2 的 list。

（4）(x :: y) :: L，不可以和 int list 匹配成功，模式(x :: y)就表示一个 list，那么模式(x :: y) :: L 就表示一个 list list，所以其不可能和 int list 匹配成功。其模式类型可以是 int list list。

（5）[x, y]，可以和 int list 匹配成功，这句话表示元素个数为 2 的 list。

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
- lists of length 2 or 3
- Non-empty lists of pairs
- Pairs with both components being non-empty lists

（1） 长度为 3 的 list 的模式为：[x, y, z]

（2） 没有这样的模式与其对应，因为我们没有一种模式能够指定 list 中的元素个数为两个不同的且具体确定的值（并且 SML 中没有模式能够对两种特定的模式做或运算成为一种合并的模式）。

（3） 非空 pair 的 list 的模式为：(x, y) :: L

（4） 一个 pair 中两个元素都是 list 的模式为：((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？

解答：通读整个程序段，整个 assmeble 函数的计算公式如下：

$$assemble(x, y) = x + trunc\big(12.4 * (y - 1)\big)$$

其中 trunc 的意思是截断（取整），并且题目中已经给 x 赋值为 3。

现在来回答题目中的提问：

- 第 4 行中 x 的绑定类型和值分别是：2 : int（即 int 类型、值为 2）
- 第 5 行中 m 的绑定类型和值分别是：12.4 : real（即 real 类型、值为 12.4）
- 第 6 行中 x 的绑定类型和值分别是：9001 : int（即 int 类型，值为 9001）
- 最后 assemble(x, 3.0)的计算如下：

$$assemble(x, 3.0) = 3 + trunc\big(12.4 * (3 - 1)\big) = 3 + 24 = 27$$

即$assemble(x, 3.0) = 27$，其值的类型为 int。

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。

解答：

（1）首先我们构造两个函数：

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

  函数要我们合并两个 list，并且合并的元组元素个数跟随两 list 中较小的个数。

  - ✓ 首先如果 L1 空，L2 任意；或者 L2 空，L1 任意，最后得到的都是空 list

✓ 其他情况下将 L1 和 L2 首元素取出，合并成一个 pair，L1、L2 剩下的 list 继续执行 zip 操作。

函数实现如下：

```
(* question 4-1 *)
(* zip: string list * int list -> (string * int) list *)
(* 分别提取string list和int list中的i号元素，组成二元组，如果长度不同按照长度小的算 *)
fun zip ([ ] : string list, L2 : int list) : (string * int) list = [ ]
    | zip (L1, [ ]) = [ ]
    | zip (x :: L1, y :: L2) = (x, y) :: zip (L1, L2)
```

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

函数要实现 zip 的逆操作，将 pair list 中的元素归还给两个 list。

✓ 首先如果 pair list 为空，则结果的两个 list 都为空

✓ 其他情况下，取出 pair list 的首元素(x, y)，将 x 归还给 string list，将归还给 int list，pair list 的其他元素继续执行 unzip 操作。

函数实现如下：

```
(* question 4-2 *)
(* unzip: (string * int) list -> string list * int list *)
(* 执行zip的反向操作，将二元组list中的元素还原为string list和int list *)
fun unzip ([ ] : (string * int) list) : string list * int list = ([ ], [ ])
    | unzip ((x , y) :: L) = let val (X, Y) = unzip L in (x :: X, y :: Y) end
```

➢ **两个函数的测试：**

```
(* test *)
val result1 = zip(["book","pen","pencil","computer"],[15,3,1,4000])
(* [("book",15),("pen",3),("pencil",1),("computer",4000)]: (string * int) list *)
val result2 = zip(["apple","banana","strawberry","watermelon"],[4,3,5])
(* [("apple",4),("banana",3),("strawberry",5)]: (string * int) list *)

val result3 = unzip result1
(* (["book","pen","pencil","computer"],[15,3,1,4000]): string list * int list *)
val result4 = unzip result2
(* (["apple","banana","strawberry"],[4,3,5]): string list * int list *)
```

```
- use "hw1.sml";
[opening hw1.sml]
val zip = fn : string list * int list -> (string * int) list
val unzip = fn : (string * int) list -> string list * int list
val result1 = [("book",15),("pen",3),("pencil",1),("computer",4000)]
  : (string * int) list
val result2 = [("apple",4),("banana",3),("strawberry",5)]
  : (string * int) list
val result3 = (["book","pen","pencil","computer"],[15,3,1,4000])
  : string list * int list
val result4 = (["apple","banana","strawberry"],[4,3,5])
  : string list * int list
val it = () : unit
```

测试的解释：

✓ result1 和 result2 是测试 zip 函数，其中 result2 的 int list 比 string list 少一个元素，可以看出结果中元素个数以 int list 为准。

✓ result3 和 result4 是测试 unzip 函数，结果是可以准确分离 int list 和 string list。

（2）回答 unzip(zip (L1, L2)) = (L1, L2)是否成立。

显然这个结论是不成立的，如果开始的时候 L1 和 L2 的元素个数是不相等的，经过先 zip 再 unzip 的变换之后，两个 list 元素个数就想等了，元素个数多的 list 的末尾元素就会被截断。

举个例子如下：

$$unzip\big(zip([\text{"book"},\text{"pen"}],[15,3,1])\big)$$

$$= unzip([(\text{"book"},15),(\text{pen},3)])$$

$$= ([\text{"book"},\text{"}pen\text{"}]),[15,3]$$

$$\neq ([\text{"book"},\text{"}pen\text{"}]),[15,3,1]$$

证明了我们的分析无误，进而进一步说明了 unzip(zip (L1, L2)) = (L1, L2)是不成立的。

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;
```

```
(* circ : real -> real *)
fun circ (r : real) : real = 2 * pi * r

(* semicirc : real -> real *)
fun semicirc : real = pie * r

(* area : real -> real *)
fun area (r : int) : real = pi * r * r
```

解答：我们逐句分析每句代码，指出其中的错误并改正：

（1） 第一句：对 pi 的赋值，没有错误。

（2） 第二句：实现 n!的计算，没有错误。（但要注意此处 n 输入初始值应该大于等于 0）

（3） 第三句：函数 f 输入 3 则得到 9，如果输入其他值只能得到 4。此处有错误，在第二行代码前应该加上‘|’。正确的代码是：

fun f (3 : int) : int = 9

| f _ = 4

（4） 第四句：计算半径为 r 的圆的周长，这句话有错误，计算类型为 real 型，而 2 为 int 型。应该由 real 型替代，为 2.0，正确代码如下：

fun circ (r : real) : real = 2.0 * pi * r

（5） 第五句，计算半径为 r 的圆的半周长，这句话中 pie 这个量的名称写错了，应该改为 pi；且在函数声明时没有声明参数 r，正确的代码如下：

fun semicirc (r : real) : real = pi * r

（6） 第六句，计算半径为 r 的圆的面积，这句话中 pi 是 real 型的，而我们将 r 声明为 int 型的显然不正确，应该将 r 声明为 real 型的，正确代码如下：

fun area (r : real) : real = pi * r * r

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);

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
```

解答：本题要分析 fib 和 fibber 的 work。

（1） fib 的 work 分析如下：

$W_{fib}(0) = O(1)$

$W_{fib}(1) = O(1)$

$$W_{fib}(n) = W_{fib}(n-1) + W_{fib}(n-2) + O(1)$$

将最后一个式子展开有：

$$W_{fib}(n) = W_{fib}(n-1) + W_{fib}(n-2) + O(1)$$

$$= (W_{fib}(n-2) + W_{fib}(n-3)) + (W_{fib}(n-3) + W_{fib}(n-4)) + (1+2) * O(1)$$

$$= O(2^n) + O(2^n) + O(1 + 2 + 2^2 + 2^3 + \cdots 2^n) * O(1)$$

$$= O(2^n) + O(2^n) + O(2^n) * O(1)$$

$$= O(2^n)$$

（2） fibber 的 work 分析如下：

$$W_{fibber}(0) = O(1)$$

$$W_{fibber}(n) = W_{fib}(n-1) + O(1)$$

将最后一个式子展开有：

$$W_{fibber}(n) = W_{fib}(n-1) + O(1) = (n-1+1) * O(1) = O(n)$$