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 可以匹配成功

\_::\_ 可以匹配成功x::(y::L) 可以匹配成功(x::y):: L匹配不成功，类型是int list list[x, y] 可以匹配成功

1. 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的模式是[x,y,z]lists of length 2 or 3没有这种模式，无法指定list长度为两个值Non-empty lists of pairs模式是（x，y）：：LPairs with both components being non-empty lists((a：：A),(b：：B))
2. 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行中的x的声明绑定的类型是int，值是2

第5行中的m的声明绑定的类型是real，值是12.4

第6行中的x的声明绑定的类型是int，值是9001

第14行表达式assemble(x, 3.0)计算的结果是27,类型为int

1. 指出下列代码的错误： (\* pi: real \*)

val pi : real = 3.14159; 没有错误

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

没有错误

(\* f : int -> int \*)fun f (3 : int) : int = 9 f \_ = 4;缺少|符号

(\* circ : real -> real \*)fun circ (r : real) : real = 2 \* pi \* r2为int，应该改为2.0

(\* semicirc : real -> real \*)fun semicirc : real = pie \* r没有声明参数

(\* area : real -> real \*)fun area (r : int) : real = pi \* r \* r声明中参数是real型，而参数列表中是int型

1. 在提示符下依次输入下列语句，观察并分析每次语句的执行结果。3+ 4; 结果为val it = 7 : int，语句是经行3和4的求值，结果为int型3 + 2.0; 结果输出错误，3是int型而2.0是real型it + 6; 结果为val it = 13 : int，语句将值为7的int与6求和val it = “hello”; 结果为val it = "hello" : string，语句将it重新赋值为字符串it + “ world”; 结果输出错误，字符串不能进行+操作,应该用^it + 5; 结果输出错误，字符串不能和int型相加val a = 5; 结果为val a = 5 : int，是声明int型变量a为5a = 6; 结果为val it = false : bool，a值为5，与6进行等式判断输出falsea + 8; 结果为val it = 13 : int，a值为5，加8后为值为13的int型变量val twice = (fn x => 2 \* x); 结果为val twice = fn : int -> int，表示有了一个名为twice，参数为int型，输出为int型的函数twice a; 结果为val it = 10 : int，将a传入函数中，得到值为10的结果let x = 1 in x end; 输出错误，x=1是bool值，赋值应该用val x=1foo; 输出错误，foo未绑定[1, “foo”]; 输出错误，list内应该为同一类型成员
2. 函数sum用于求解整数列表中所有整数的和，函数定义如下：

(\* sum : int list -> int \*)(\* REQUIRES: true \*)(\* ENSURES: sum(L) evaluates to the sum of the integers in L. \*)fun sum [ ] = 0 | sum (x ::L) = x + (sum L); 完成函数mult的编写，实现求解整数列表中所有整数的乘积。(\* mult : int list -> int \*)(\* REQUIRES: true \*)(\* ENSURES: mult(L) evaluates to the product of the integers in L. \*)fun mult [ ] = 1 | mult (x ::L) = x\*(mult L)；

1. 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。

fun zip([]:string list,[]:int list)=[]

|zip([],b::L2)=[]

|zip(a::L1,[])=[]

|zip(a::L1,b::L2)=(a,b)::zip(L1,L2);

（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。

fun unzip([])=([],[])

|unzip((a,b)::L)=

let

val(L1,L2) =unzip L

in

(a::L1,b::L2)

end;

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。

不是都成立，当L1和L2不等长时，zip会丢失一部分数据，而再次unzip时不会将丢失的数据补回，从而不相等。

1. 完成如下函数Mult: int list list -> int的编写,该函数调用mult 实现int list list中所有整数乘积的求解。  
    (\* mult : int list list -> int \*)(\* REQUIRES: true \*)(\* ENSURES: mult(R) evaluates to the product of all the integers in the lists of R. \*) fun Mult [ ] = 1 | Mult (r :: R) = mult(r)\*Mult(R);
2. 函数mult’定义如下，试补充其函数说明，指出该函数的功能。(\* mult’ : int list \* int -> int \*)(\* REQUIRES: true \*)(\* ENSURES: mult’(L, a)将函数中每一个元素都乘以a\*) fun mult’ ([ ], a) = a | mult’ (x :: L, a) = mult’ (L, x \* a);

利用mult’定义函数Mult’ : int list list \* int -> int，使对任意整数列表的列表R和整数a，该函数用于计算a与列表R中所有整数的乘积。该函数框架如下所示，试完成代码的编写。fun Mult’ ( [ ], a) = a | Mult’ (r::R, a) = Mult’(R,mult’(r,1)\*a);

1. 编写递归函数square实现整数平方的计算，即square n = n \* n。 要求：程序中可调用函数double，但不能使用整数乘法（\*）运算。(\* double : int -> int \*)(\* REQUIRES: n >= 0 \*)(\* ENSURES: double n evaluates to 2 \* n.\*)fun double (0 : int) : int = 0 | double n = 2 + double (n - 1)fun square(0:int):int =0

|square n=square(n-1) + double(n)-1;