

**函数式编程实验报告**

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

姓名：

班级：

计算机科学与技术学院

2020 年 4月 19 日

**目 录**

[1. 实验一 3](#_Toc38122012)

[**1.1.** **实验代码** 3](#_Toc38122013)

[**1.2.** **实验结果** 4](#_Toc38122014)

[2. 实验二 5](#_Toc38122015)

[**2.1.** **实验代码** 5](#_Toc38122016)

[**2.2.** **实验结果** 6](#_Toc38122017)

[3. 实验三 7](#_Toc38122018)

[**3.1.** **实验代码** 7](#_Toc38122019)

[**3.2.** **实验结果** 9](#_Toc38122020)

[4. 实验心得 11](#_Toc38122021)

# 实验一

* 1. **实验代码**

print("1.2\n");

fun mult [ ] = 1

| mult (x::l) = x \* mult (l);

val list1 =[1,2,3,4];

val res1 = mult(list1);

print("\n1.3");

fun Mult [ ] = 1

| Mult (r::R) = mult(r)\*Mult(R);

val list2 =[[1,2,3],[4,5]];

val res2 = Mult(list2);

print("\n1.4");

fun mult' ([ ], a) = a

| mult' (x :: L, a) = mult' (L, x \* a);

fun Mult' ([ ], a) = a

| Mult' (r::R, a) = Mult(R) \* mult'(r,a);

val list3 = ([[1,2,3],[4,5]],6);

val res3 = Mult'(list3);

print("\n1.5\n");

fun double (0 : int) : int = 0

| double n = 2 + double (n - 1);

fun square(0:int):int =0

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

val num = 5;

val res4 = square(num);

print("\n1.6\n");

fun divisibleByThree(x:int):bool =

if x=0 then true

else if x=1 then false

else if x=2 then false

else divisibleByThree(x-3);

val num = 14;

val res5 = divisibleByThree(num);

print("\n1.7\n");

fun evenP (0 : int) : bool = true

| evenP 1 = false

| evenP n = evenP(n - 2);

val num = 13;

val res6 = evenP(num);

* 1. **实验结果**

实验结果如图：

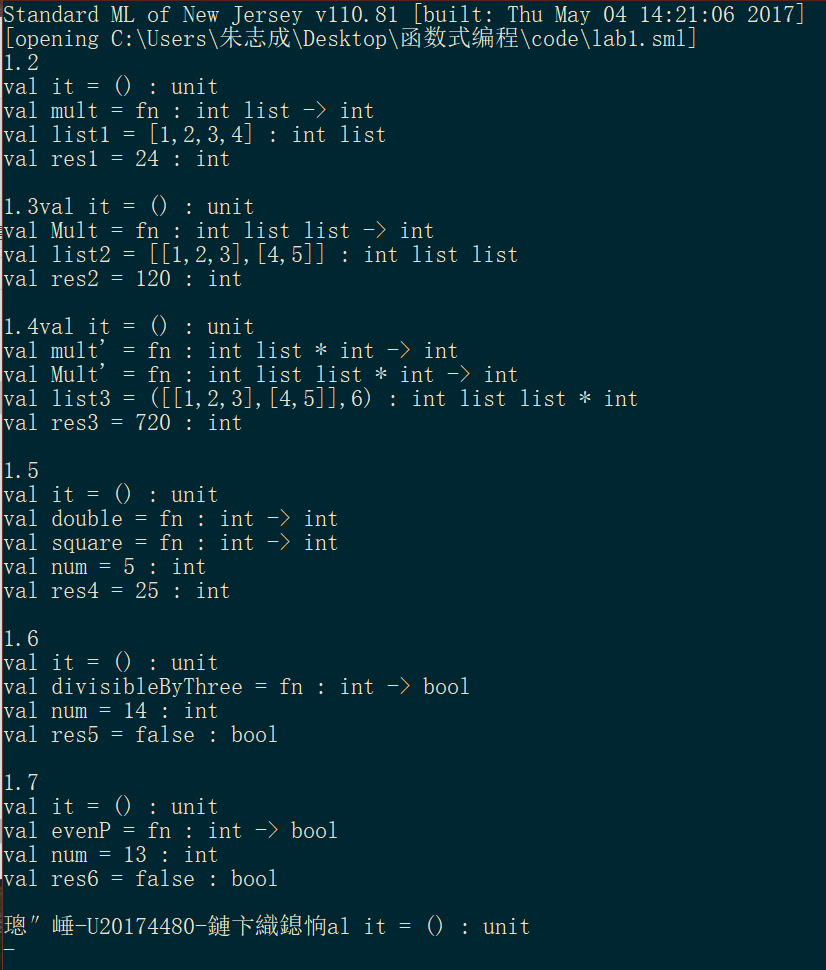


图1.1 lab1\_res

# 实验二

* 1. **实验代码**

print("lab2.1\n");

fun reverse [] = []

|reverse (a::L) = (reverse L) @ [a] ;

val exp1 = [1,2,3];

reverse(exp1);

fun reverse'(L:int list): int list =

let

fun helper(L:int list , A:int list):int list =

case L of

[] => A

| x::R => helper(R, x::A)

in

helper(L,[])

end;

reverse(exp1);

print("\nlab2.2\n");

fun interleave(A : int list , B : int list) : int list =

case (A , B) of

(\_,[]) => A

|([],\_) => B

|(a::L1,b::L2) => a::b::interleave(L1,L2);

val exp21 = [1,2,3];

val exp22 = [4,5,6];

interleave(exp21,exp22);

print("\nlab2.3\n");

datatype tree = Empty | Node of tree \*int \*tree;

fun split(l: int list) =

let

val m = (length l) div 2

val L = (List.take (l,m))

val x :: R = (List.drop (l,m))

in

(L, x, R)

end;

fun listToTree([]:int list):tree = Empty

| listToTree(L)=

let

val (l,x,r) = split(L)

in

Node(listToTree(l),x,listToTree(r))

end;

val exp3=[1,2,3,4,5];

listToTree(exp3);

print("\nlab2.4\n");

fun revT (Empty : tree):tree = Empty

| revT (Node(l,x,r)) = Node (revT (r), x , revT (l));

val exp4=listToTree([1,2,3]);

revT(exp4);

print("\nlab2.5\n");

fun binarySearch(Empty:tree, i: int): bool = false

|binarySearch(Node(l,x,r), i) =

case Int.compare(x,i) of

GREATER => binarySearch(l,i)

| EQUAL => true

| LESS => binarySearch(r,i)

val exp5 = listToTree([1,2,3,4,5]);

binarySearch(exp5,5);

* 1. **实验结果**

实验结果如图：

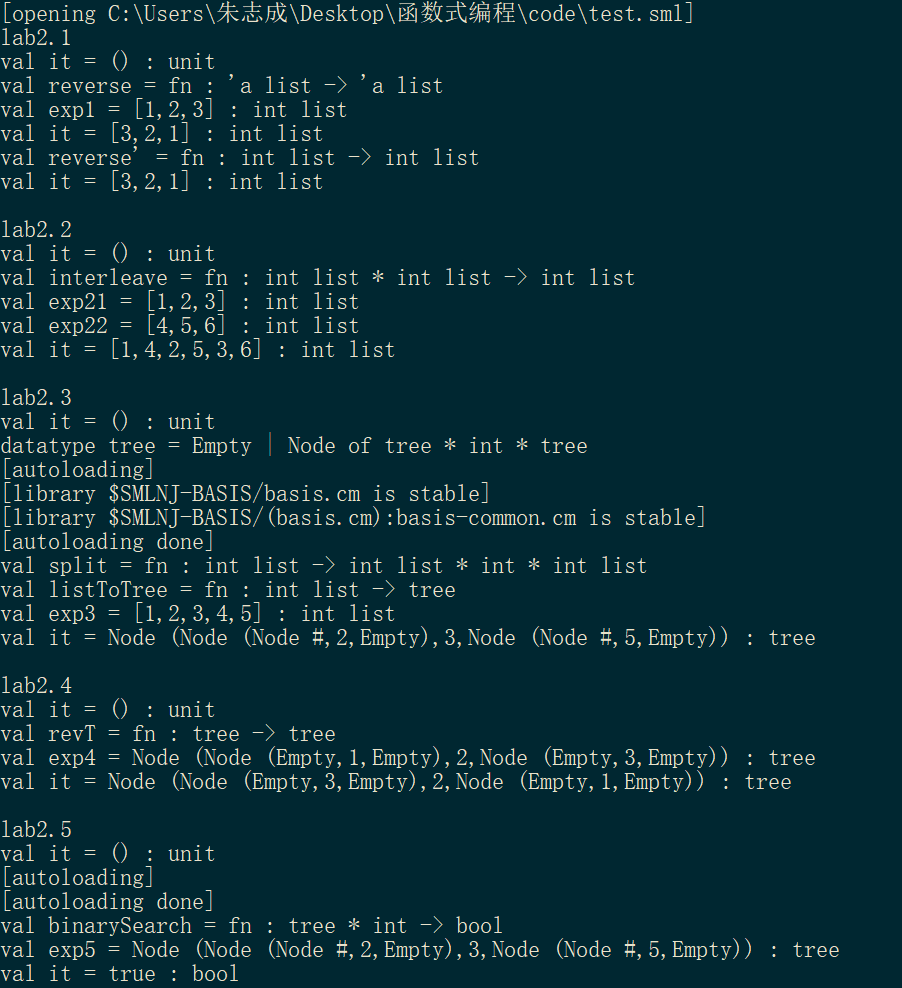


图2.1 lab2\_res

# 实验三

* 1. **实验代码**

print("LAB3.1\n");

fun sqrt(x:int):int = x \* x;

fun thenAddOne( (f:int->int), (x:int) ):int = f x + 1;

val exp1 = 5;

thenAddOne(sqrt,exp1);

print("\nLAB3.2~3.3\n");

fun mapList (f: 'a -> 'b, [] : 'a list) : 'b list = []

|mapList (f, x::L) = (f x)::(mapList (f, L));

fun mapList' (f : 'a -> 'b) ([] : 'a list) : 'b list = []

|mapList'(f)(x::L) = (f x)::(mapList' f L);

val exp2 = [1,2,3,4,5];

mapList(sqrt,exp2);

val exp3 = [1,3,5,7,9];

mapList' sqrt exp2;

print("\nLAB3.4\n");

fun findOdd([] : int list): int option = NONE

| findOdd(x::L) = if (x mod 2)=1 then SOME x else findOdd L;

val exp4 = [1,8,3,4,6];

findOdd(exp4);

print("\nLAB3.5\n");

fun subsetSumOption ([] : int list, n : int) : int list option = NONE

| subsetSumOption (l , 0 ) = SOME []

| subsetSumOption (x::L, sum) =

if subsetSumOption(L, sum-x) = NONE

then

subsetSumOption(L, sum)

else

SOME ( x::( valOf( subsetSumOption(L, sum-x) ) ) );

val exp5 = [1,2,3,4,5];

val tar = 7;

subsetSumOption(exp5,tar);

print("\nLAB3.6\n");

fun exists (p : 'a -> bool) (l : 'a list) : bool =

foldl (fn (a,b) => p a orelse b) false l;

fun forall (p : 'a -> bool) (l : 'a list) : bool =

foldl (fn (a,b) => p a andalso b) true l;

fun p(n:int):bool= if n>4 then true else false;

val exp6\_1 = [1,2,3,4];

exists p exp6\_1;

val exp6\_2 = [5,6,7,8];

forall p exp6\_2;

print("\nLAB3.7\n");

datatype 'a tree = Empty | Node of 'a tree \* 'a \* 'a tree;

fun split(l: int list) =

let

val m = (length l) div 2

val L = (List.take (l,m))

val x :: R = (List.drop (l,m))

in

(L, x, R)

end;

fun listToTree([]:int list):int tree = Empty

| listToTree(L) =

let

val (l,x,r) = split(L)

in

Node(listToTree(l),x,listToTree(r))

end;

fun treeFilter (p : 'a -> bool ) (Empty) : 'a option tree = Empty

| treeFilter (p) (Node (l, x, r)) =

let

val y = if p x then SOME x else NONE

in

Node (treeFilter p l, y, treeFilter p r)

end;

val exp7=listToTree([1,2,3,4,5]);

treeFilter p exp7;

* 1. **实验结果**

实验结果如图:

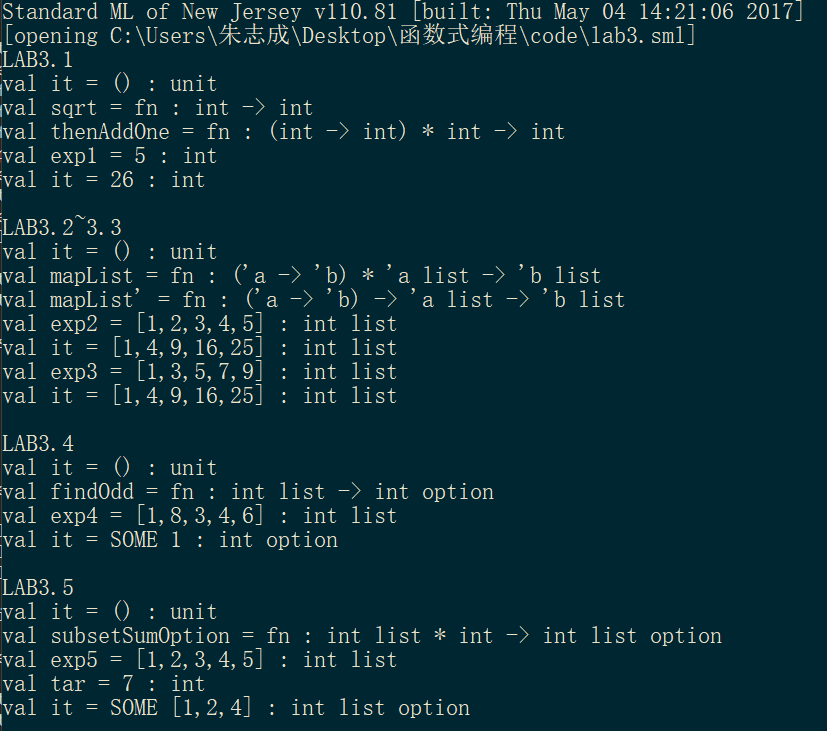


图3.1 Lab3\_res1

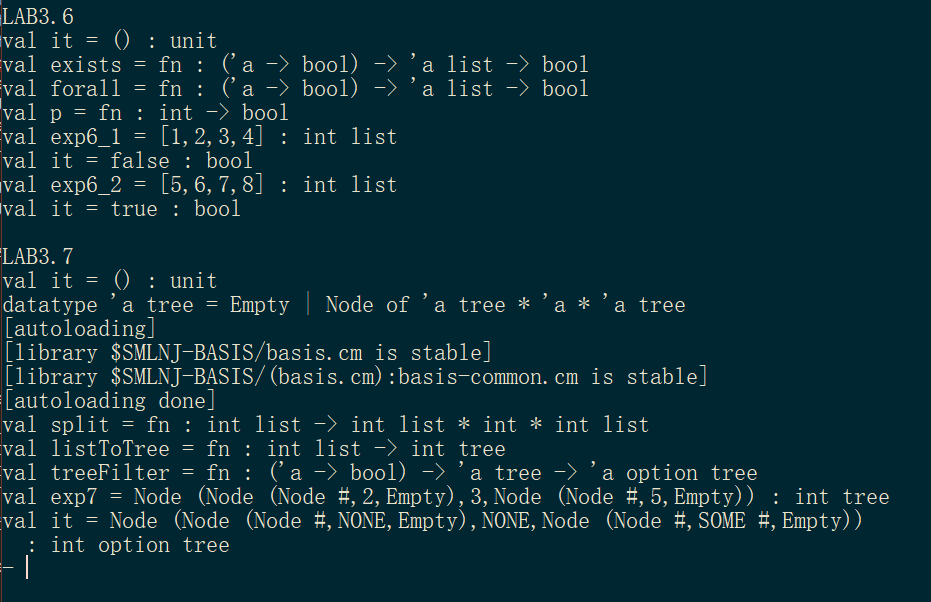


图3.1 Lab3\_res2

# 实验心得

刚开始接触的时候比较艰难，因为函数式编程不太符合自己的编程习惯。可能和网课的学习方式有关，每次看完视频后对于相关知识的理解总感觉差了一些，印象也不太深刻，经过后续多次看书和课件才有了较好的掌握。

实验的难度较为适中，但是在后两次实验中也碰到了一些困难，好在网上有较多的参考资料，许多思路不清晰的地方对照着代码看都能较好的理解，完成实验。

通过此次课程，我觉得个人提升最大的地方在于对于递归过程的理解和分析，包括在模式匹配的相关内容中对于递归出口有了一个较好的掌握。

目前我对于函数式编程的理解还比较粗浅，我认为函数式编程就是把运算过程尽量写成一系列嵌套的函数调用的过程，代码很简洁，有些像自然语言，比较容易理解，像是一种编程的范式。

这门课程给我带来了比较大的收获。