7.2.4：下面是两个C语言函数f和g的概述：

```
int f ( int x ) { int i; … return i + 1; … }
int g ( int y ) { int j; … f ( j + 1 ); … }
```

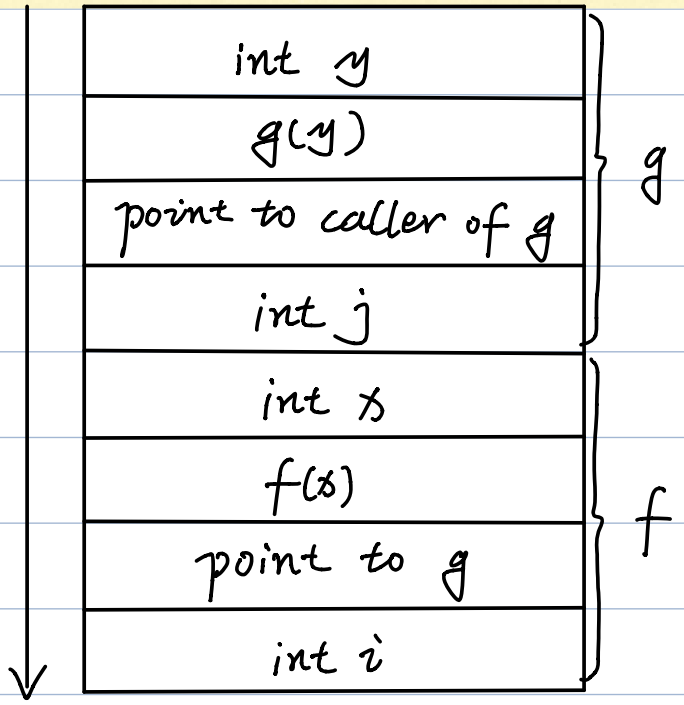
函数g调用f。画出在g调用f而f即将返回时，运行时刻栈中从g的活动记录开始的顶端部分。你可以只考虑返回值、参数、控制链以及存放局部数据的空间。你不用考虑存放的机器状态，也不用考虑没有在代码中显示的局部值和临时值。但是你应该指出：

1) 哪个函数在栈中为各个元素创建了所使用的空间?

2) 哪个函数写入了各个元素的值?

3) 这些元素属于哪个活动记录?

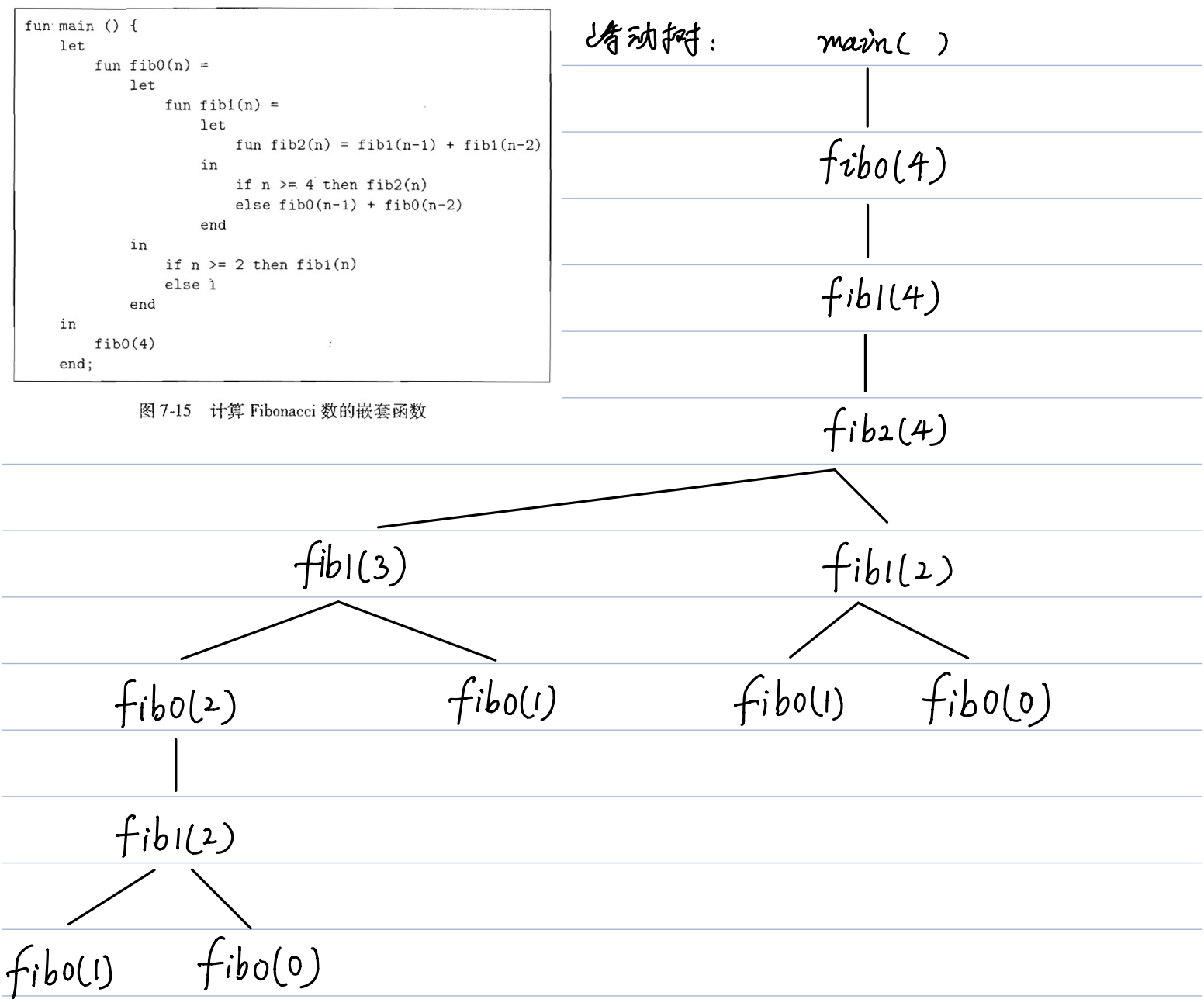
解：



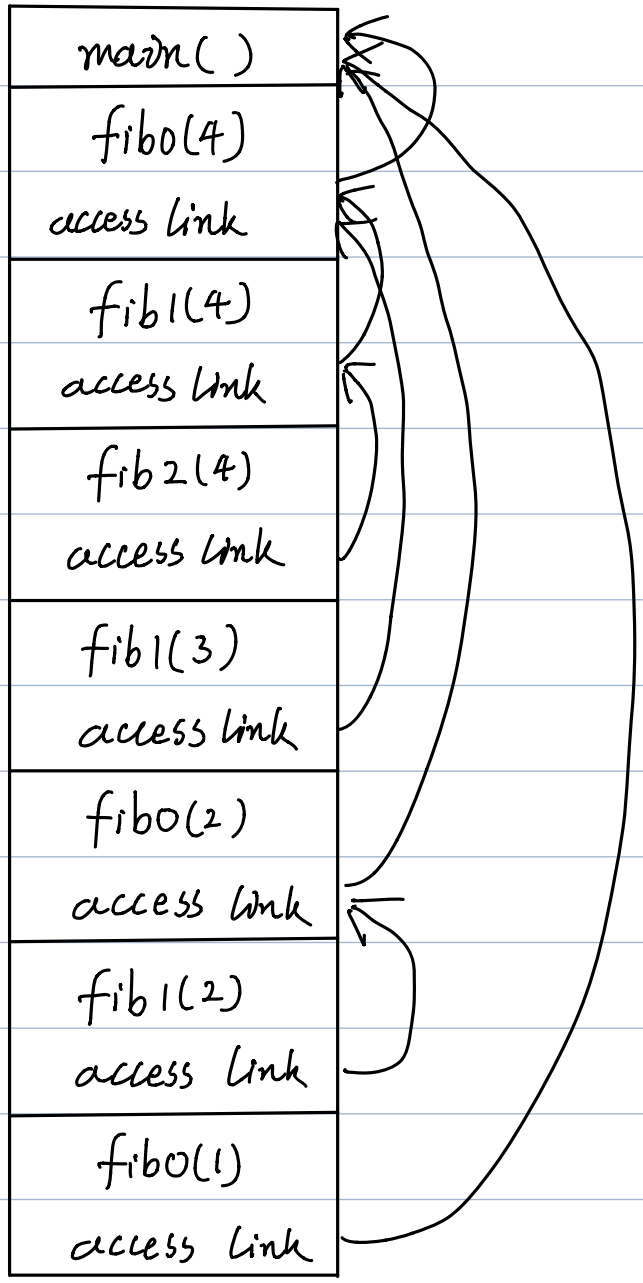
7.3.1：图7-15（见下页）中给出了一个按照非标准方式计算Fibonacci数的ML语言的函数main。函数fib0将计算第n个Fibonacci数（n≥0）。嵌套在fib0中的是fib1，它假设n≥2并计算第n个Fibonacci数。嵌套在fib1中的是fib2，它假设n≥4。请注意，fib1和fib2都不需要检查基本情况。我们考虑从对main的调用开始，直到（对fib0(1)的）第一次调用即将返回的时段，请描述出当时的活动记录栈，并给出栈中的各个活动记录的访问链。

```
fun main () {
    let
        fun fib0(n) =
            let
                fun fib1(n) =
                    let
                        fun fib2(n) = fib1(n-1) + fib1(n-2)
                    in
                        if n >= 4 then fib2(n)
                        else fib0(n-1) + fib0(n-2)
                    end
            in
                if n >= 2 then fib1(n)
                else 1
            end
    in
        fib0(4)
    end;
```

图 7-15　计算 Fibonacci 数的嵌套函数



堆动栈：


main( )
fibo(4)
access link
fib1(4)
access link
fib2(4)
access link
fib1(3)
access link
fibo(2)
access link
fib1(2)
access link
fibo(1)
access link

7.3.2：假设我们使用display表来实现下图中的函数。请给出fib0(1)的第一次调用即将返回时的display表。同时指明那时在栈中的各个活动记录中保存的display表条目。

display表：