Solution11

191220008 陈南疃

概念题

1、什么是函数式编程？它有什么优缺点。

- 函数式程序设计（functional programming）是指把程序组织成一组数学函数，计算过程体现为基于一系列函数应用（把函数作用于数据）的表达式求值。
- 函数也被作为值（数据）来看待，即函数的参数和返回值也可以是函数。
- 基于的理论是递归函数理论和lambda演算。

优点：

- 代码简洁，开发快速：函数式编程大量使用函数，减少了代码的重复，因此程序比较短，开发速度较快。
- 接近自然语言，易于理解：函数式编程的自由度很高，可以写出很接近自然语言的代码。
- 更方便的代码管理：函数式编程不依赖、也不会改变外界的状态，只要给定输入参数，返回的结果必定相同。因此，每一个函数都可以被看做独立单元，很有利于进行单元测试（unit testing）和除错（debugging），以及模块化组合。
- 易于"并发编程"：函数式编程不需要考虑"死锁"（deadlock），因为它不修改变量，所以根本不存在"锁"线程的问题。不必担心一个线程的数据，被另一个线程修改，所以可以很放心地把工作分摊到多个线程，部署"并发编程"（concurrency）。
- 代码的热升级：函数式编程没有副作用，只要保证接口不变，内部实现是外部无关的。所以，可以在运行状态下直接升级代码，不需要重启，也不需要停机。

缺点：

严重耗费在CPU和存储器资源

- 早期的函数式编程语言实现时并无考虑过效率问题。
- 有些非函数式编程语言为求提升速度，不提供自动边界检查或自动垃圾回收等功能

惰性求值亦为语言如Haskell增加了额外的管理工作。

2、什么是尾递归和尾递归优化?

尾递归：

- 递归调用是递归函数的最后一步操作，并且，return语句不能包含表达式

尾递归优化：

- 由于递归调用是本次调用的最后一步操作，因此，递归调用时可重用本次调用的栈空间。
- 可以自动转成迭代。

3、C++中的Filter、Map和Reduce操作各自是什么含义。

Filter（过滤）：把一个集合中满足某条件的元素选出来，构成一个新的集合。

Map（映射）：分别对一个集合中每个元素进行某种操作，结果放到一个新集合中。

Reduce（规约）：对一个集合中所有元素进行某个操作后得到一个值。

4、C++是如何实现Currying操作的？并阐述一下该操作的重要性。

Currying（柯里化）：把接受多个参数的函数变换成接受单一参数（原函数的第一个参数）的函数，该函数返回一个接收剩余参数的函数。

如何实现：

在C++中，通过用 lambda 表达式返回函数的方式实现 Currying 操作（或者用bind）。

重要性：

① 解决不想写或者不好写闭包的问题。其实就是配合其他函数的调用场景。

② 科里化把函数的层层调用和封装，变成了状态的转换。函数每调用一次，生成一个新的函数，即相当于迁移至了一个新的状态上。于是程序代码更加接近状态机，也会有助于程序结构的梳理。

③ 柯里化函数在使用上更为简洁，简化语法设计，可以更好地处理和抽象代码的逻辑。

④ 利用柯里化，我们可以固定住其中的部分参数，在调用时这个参数就不需要再次传递，即参数复用。

⑤ 使用通用的实现 currying 方法来实现加法函数，即可看到延迟计算的效果。

⑥ 使函数具有部分求值的能力。

⑦ 可以更好的实现偏函数。

编程题

1、使用尾部递归实现算法来找到二叉树中最长的Z型路径长度。

```
#include <iostream>

using namespace std;

struct Node {
    Node* left;
    Node* right;
    Node() { left = NULL; right = NULL; }
```

```
};

int z_max = 0;

/*left: direction = 0, right: direction = 1*/
void DFS(Node* p, bool direction, int z_length) {
    if (z_length > z_max)
        z_max = z_length;
    if (direction == 0) {
        if (p->left)
            DFS(p->left, 1, z_length + 1);
        if (p->right)
            DFS(p->right, 0, 1);
    }
    else {
        if (p->right)
            DFS(p->right, 0, z_length + 1);
        if (p->left)
            DFS(p->left, 1, 1);
    }
}

int main()
{
    Node* root;
    /*Create a 2-Tree with the root named "root"*/
    DFS(root, 0, 0);
    DFS(root, 1, 0);
    cout << z_max << endl;
    return 0;
}
```

2、函数求导数（currying）。

```
#include <iostream>
#include <vector>
#include <functional>
#include <cmath>
#include <algorithm>

using namespace std;
using namespace std::placeholders;

double derivative(double x, double d, double (*f)(double)) {
    return (f(x) - f(x - d)) / d;
}

auto bind_derivative(double x, double d) {
    return std::bind(derivative, x, d, _1);
}

auto bind_derivative(double x) {
    return [x](double d) {return std::bind(derivative, x, d, _1); };
```

```
}

int main() {
    std::vector<double (*)(double)> funcs = { sin, cos, tan, exp, sqrt, log,
log10 };
    // 目标函数
    auto d1 = bind_derivative(1, 0.000001); // 在x=1处求导数的函数d1
    auto d2 = bind_derivative(1)(0.000001); // 在x=1处求导数的函数d2
    std::vector<double> result1, result2;
    std::transform(funcs.begin(), funcs.end(), std::back_inserter(result1), d1);
    std::transform(funcs.begin(), funcs.end(), std::back_inserter(result2), d2);
    // result1的结果与result2的结果相同
    return 0;
}
```