# Assignment 11

**FY Wang**

## 简答题

1. 什么是函数式编程?

函数式程序设计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的表达式求值。函数也被当作值来看待，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。

函数式设计的基本特征

- 纯函数：以相同的参数调用一个函数总得到相同的值。
- 没有状态，计算不改变已有的数据，而是产生新的数据
- 函数也是值
- 递归式主要的控制结构，重复操作不采用迭代的方式实现
- 表达式的惰性求值需要时才计算
- 潜在的并行性。

2. 什么是尾递归?

递归是实现重复操作，而不是用循环迭代的方式。

尾递归是递归调用是递归函数的最后一步操作。

3. C++ 中Filter 、 Map 、Reduce操作 各自是什么含义?

Filter： 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：分别对一个集合中的每个元素进行某种操作，将结果放到一个新的集合当中。

Reduce：对一个集合中所有元素进行某个操作后得到一个值。

bind: 给一个函数的某些参数指定固定的值，返回一个含有一个为指定参数所构成的函数。

curry：把接受多个参数的函数变换成接受单一参数的函数，该函数返回一个接受剩余参数的函数。

4. currying 在C++中如何实现，并且有什么意义?

C++ 中使用bind来实现currying。

currying的一个重要意义在于起降函数完全变成了接受一个参数，返回一个值得固定形式，对于讨论和优化会更加的方便。同时简化了函数式编程的复杂性，是的函数式编程更加的优雅。

## 编程题

1.

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
```

```
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x),
left(left), right(right) {}
 * };
 */



class Solution {
private:
    int ans;
    void dp(TreeNode *root, bool isleft, int depth)
    {
        if (root==nullptr) return ;
        ans=ans>depth+1?ans:depth+1;
        if (isleft)
        {
            dp(root->right,false, depth+1);
            dp(root->left,true,0);
        }
        else
        {
            dp(root->left,true,depth+1);
            dp(root->right,false,0);

        }
    }

public:
    int longestZigZag(TreeNode* root) {
        this->ans=0;
        dp(root->left, true, 0);
        dp(root->right,false,0);
        return ans;
    }
};
```

2.

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

//使用两个using namespace 会自动地选择?
using namespace std;
using namespace std::placeholders;
/* 求导数函数，对某个函数f在点x0处求得导数
* f'(x0) = (f(x0)-f(x0-d))/d
* params:
* x: x0
* d: d
* f: f
```

```
*/

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



function<double(double(*)(double))> bind_derivative(double x,double d)
{
    return bind(derivative,x,d,_1);
}

// 或者直接用auto 但是要用--std=c++1y.

function<function<double(double(*)(double))>(double)> bind_derivative(double x)
{
    return [x](double d)->function<double(double(*)(double))>{return bind(derivative,x,d,_1);};
}

int main() {
 std::vector<double (*)(double)> funcs = {sin, cos, tan, exp, sqrt, log, log10};
// 目标函数
 //bind_derivative接受两个参数，同时返回一个接受一个参数的函数。
 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
 for_each(result1.begin(),result1.end(),[](double a)->void{cout <<a<<endl;});
 for_each(result2.begin(),result2.end(),[](double a)->void{cout <<a<<endl;});

 return 0;
}
```