HW11

181220076 周韧哲

一. 概念题

1. 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（把函数作用于数据）的表达式求值。优点：容易阅读，强调纯函数，无副作用，不涉及状态改变，测试和调试很方便。缺点：性能比命令式编程差，不适合处理可变状态，不适合IO操作等。
2. 若一个函数中所有递归形式的调用都出现在函数的末尾，则这个递归函数是尾递归的。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，且可以自动转成迭代。
3. filter：按照某个规则，将符合规则的留下，其余删除。map：将某个函数作用于每个元素上。reduce：对数据进行一个连续的操作，它的函数是双目运算符，如+，x等。
4. c++借助bind实现了currying操作，currying能将一个有n个参数的函数转换成n个只有1个参数的函数，能够实现多参函数，其重要意义在于可以把函数完全变成「接受一个参数；返回一个值」的固定形式，这样对于讨论和优化会更加方便。

二. 编程题

1.
```
struct Node {
    int val;
    Node *left;
    Node *right;
    Node() : val(0), left(nullptr), right(nullptr) {}
    Node(int x) : val(x), left(nullptr), right(nullptr) {}
    Node(int x, Node *left, Node *right) : val(x), left(left),
right(right) {}
};

void tail_recursion(Node *node, bool left, int depth, int &res){
    if(node==nullptr) return ;
    res = res>depth+1?res:depth+1;
    if(left){
        tail_recursion(node->right, false, depth+1, res);
        tail_recursion(node->left, true, 0, res);
    }else{
        tail_recursion(node->left, true, depth+1, res);
        tail_recursion(node->right, false, 0, res);
    }
}

int longest_z(Node *root){
    int res = 0;
    tail_recursion(root->left, true, 0, res);
    tail_recursion(root->right, false, 0, res);
    return res;
}

```

2.
```
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);
}

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() {
    vector<double (*)(double)> funcs = {sin, cos, tan, exp, sqrt, log, log10};
    auto d1 = bind_derivative(1, 0.000001);
    auto d2 = bind_derivative(1)(0.000001);
    vector<double> result1, result2;
    transform(funcs.begin(), funcs.end(), back_inserter(result1), d1);
    transform(funcs.begin(), funcs.end(), back_inserter(result2), d2);
    for(int i=0;i<result1.size();i++){
        if(result1[i]!=result2[i])
            assert(0);
    }
    return 0;
}
```