概念题

1. 请简述函数式程序设计的概念，函数式程序设计有哪些基本特征?
   指把程序组织成一组函数，计算过程体现为一系列的函数应用（函数作用于数据）。
   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。
      “纯”函数（引用透明）：以相同的参数调用一个函数总得到相同的值。（无副作用）
      没有状态（数据不可变）：计算不改变已有数据，而是产生新的数据。（无赋值操作）
      函数也是值（first-class citizen）：函数的参数和返回值都可以是函数。（高阶函数）
      表达式的惰性（延迟）求值（Lazy evaluation）：需要的时候才计算。
      潜在的并行性。

2. 简述逻辑式程序设计的概念，并说明它有哪些特征。
   程序由一组事实和一组推理规则构成，在事实基础上运用推理规则来实施计算。
   基于的理论是谓词演算。
      数据（事实和规则）就是程序。
      计算（匹配、搜索、回溯）由实现系统自动完成。

3. 如何理解函数式和逻辑式程序中的数据?
   函数式程序中函数也被作为数据看待，即函数的参数和返回值也可以是函数。
      数据具有不可变性，计算不改变已有数据，而是产生新的数据。
   逻辑式程序中数据（事实和规则）就是程序，运用推理规则来实施计算。

编程题

1. 两小题分别用命令式程序设计和函数式程序设计来实现，然后比较两者的区别。
   求两个正整数的最大公约数；求一个非负整数的阶乘。

```
#include <iostream>
using namespace std;

int gcd(int a, int b) {
      if (a % b == 0)
            return b;
      else
            return gcd(b, a % b);
}

int factorial(int n) {
      if (n == 0 || n == 1)
            return 1;
      else
            return n * factorial(n - 1);
}
```

```
int fact_tail(int n, int m) {
	if (n == 0 || n == 1)
		return m;
	else
		return fact_tail(n - 1, m * n);
}

int main()
{
	int a, b, c;
	cin >> a >> b;
	c = a % b;
	while (c) {
		a = b;
		b = c;
		c = a % b;
	}
	cout << b << endl;
	cout << gcd(a, b) << endl;
	int n, m = 1;
	cin >> n;
	for (int i = 1; i <= n; i++)
		m *= i;
	cout << m << endl;
	cout << factorial(n) << endl;
	cout << fact_tail(n, 1) << endl;
	return 0;
}
```

函数式程序设计比命令式程序设计更具有可读性，封装得更好，可以用递归简化计算。

2. 运用映射、规约和柯里化基本技术，分别用非函数式和函数式程序设计来实现。
   将给定的字符串中的小写字母改写成大写，结果保存在另一个字符串中。

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

char upper(char ch) {
	if (ch >= 'a' && ch <= 'z')
		return ch -= 32;
	else
```

```
        return ch;
}

string s_upper(string s) {
    string s_new;
    s_new.resize(s.size());
    transform(s.begin(), s.end(), s_new.begin(), upper);
    return s_new;
}

int main() {
    string s_old, s_new;
    cin >> s_old;
    s_new.resize(s_old.size());
    for (int i = 0; i < s_old.size(); i++) {
        if (s_old[i] >= 'a' && s_old[i] <= 'z')
            s_new[i] = s_old[i] - 32;
        else
            s_new[i] = s_old[i];
    }
    cout << s_new << endl;
    cout << s_upper(s_old) << endl;
    return 0;
}
```

计算并返回一个整型数组的累乘结果。

```
#include <iostream>
#include <vector>
#include <numeric>
using namespace std;

int multiply(vector <int> nums) {
    int product = accumulate(nums.begin(), nums.end(), 1,
        [](int a, int x) { return a * x; });
    return product;
}

int main()
{
    int temp;
    vector <int> nums;
    for (int i = 0; i < 10; i++) {
        cin >> temp;
```

```
            nums.push_back(temp);
        }
        int product = 1;
        vector<int>::iterator iter;
        for (iter = nums.begin(); iter != nums.end(); iter++) {
            product *= *iter;
        }
        cout << product << endl;
        cout << multiply(nums) << endl;
        return 0;
}
```

函数接收两个部分，返回一个完整的 URL。改写函数使之柯里化，以提高适用性。

```
#include <iostream>
#include <string>
#include <functional>
using namespace std;

string make_url(string protocol, string latter) {
        return protocol + latter;
}

function<string(string)> make_url_curry(string protocol) {
        return [protocol](string latter) { return make_url(protocol, latter); };
}

int main() {
        string protocol, latter;
        cin >> protocol >> latter;
        cout << make_url(protocol, latter) << endl;
        function < string(string)> make_with = make_url_curry("https://");
        cout << make_with(latter) << endl;
        return 0;
}
```