上海财经大学《程序设计基础》上机考试卷(A)

参考答案及评分标准

（ 2021 至 2022 学年 第 1 学期 ）

1. (循环控制。 15 分)

设计程序计算 Riemann-zeta 函数 ζ(s) 在自变量 s 为实数且大于 1 时候的近似值。ζ(s) 定义如下：

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \dots$$

当 $\frac{1}{n^s} < 10^{-6}$ 时，循环结束。

要求：当输入 s 的值小于或等于 1 时，有报错提示，例如：Invalid value of s!
当输入的 s 大于 1 时，在屏幕上输出计算的结果。

```
#include <iostream>
#include <cmath>
using namespace std;
/*
   本题中可能会使用的库函数说明如下：
      double pow(double x, double y);
   pow 函数用来求 x 的 y 次幂（次方）。
   例如： double a = pow(2,7); // 将 2^7 作为初始化值定义变量 a。
*/

int main()
{
   Double s, zetaS;
   cout<< “Please input the value of s :”;
   cin>>s;
   if(s<=1){cout<<"Invalid value of s!"<<endl;}        ----------------------------------------4 分
      else {
            for(int i=1;pow(i,-s)>=1e-6;i++){   ---------------------------------------- 4 分
                  zetaS=zetaS+pow(i,-s);        ----------------------------------------------------4 分
            }
            cout << "Value of Riemann-zeta function is "<< zetaS<<endl; ----------------3 分
         }
```

```
    return 0;
}
```

2. (函数调用。 15 分)

编写名为 counting 的函数，实现如下功能：统计数组元素中的奇数个数，偶数个数，以及找到最大的元素。

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

int main()
{
    int odd=0, even=0, a[10] = {1,3,4,8,9,7,8,14,2,5};
    int M=0;
    counting(a, odd, even, M);
    cout<<"奇数有"<< odd << "个," <<"偶数有" << even << "个."<<endl;
    cout<< "最大值为"<< M <<endl;
    return 0;
}
void counting(int*a, int&odd, int&even, int&M) -------10 分，每个分量 2 分，void 2 分
{

      for(int i=0;i<10;i++){ ------------------------------------1 分
                if(a[i]%2==0)even+=1;   ----------------------------1 分
                else odd+=1;    ----------------------------------1 分
                if(a[i]>M)M=a[i]; ----------------------------------2 分
      }
}
```

3. (数组与字符串。20 分)

编写程序，统计字符串数组中数字字符和英文字母字符的个数。定义结构体 CharCounter 用于保存数字字符和英文字母（不区分大小写）字符的个数。补充代码，实现 countChars 函数，统计一个字符串中的字符个数。

```
/*
    本题中可能会使用的库函数说明如下：
    bool isdigit(char c)函数能判断一个字符是否为一个数字字符。
    bool isalpha(char c)函数能判断一个字符是否为一个字母字符。
    int tolower(char c)函数能将大写字母转化为小写字母。
*/
#include <iostream>
```

```
#include <cstring>
#include <cctype>

using namespace std;

const int NUM = 3;
const int ALPHABETNUM = 26;

struct CharsCounter{
    unsigned int numericChars; //  数字字符个数
    unsigned int alphabet[ALPHABETNUM]; // 26 个英文字母字符的个数
};

CharsCounter cc={0,{0}}; //  定义并初始化字符计数器结构体变量 cc
void countChars(char str[]); //  统计字符串中的字符个数

int main()
{
    char myStrings[NUM][100]= {"We have a history of 500000 years.",
                               "Confucius was born 2500 years ago.",
                               "In 1949, New China was established."};

    for(int i = 0; i < NUM; i++)
        countChars(myStrings[i]);

    cout << "myStrings has " << cc.numericChars << " numeric characters." << endl;

    cout << "myStrings has English alphabets:" << endl;
    for(int i = 0; i < ALPHABETNUM; i++)
        cout << '\t' << char('a' + i) << " - " << cc.alphabet[i] << endl;

    return 0;
}

void countChars(char str[])
{
    for(int i = 0; str[i]!='\0'; i++)              // 6 分
    {
        if(isdigit(str[i]))                        //   4 分
            cc.numericChars++;
        else
            if(isalpha(str[i])){                   //   4 分
                cc.alphabet[tolower(str[i]) - 'a']++;     // 6 分
            }
```

```
    }
}
```

4. (指针与动态数组，15 分)

冒泡排序是大家非常熟悉的排序算法，现要求完全使用指针操作访问数组，来实现如下的冒泡排序函数

```
void bubbleSort(T *array_start, T *array_end);
```

其中，T 是一个模板类型；array_start 代表了数组的起始地址，array_end 代表了数组的结束地址，组成一个闭区间。该函数将对 array_start 和 array_end 之间的数据进行排序。

提示：和我们现实中的地址类似，数组的地址也是连续的，地址也可以比较大小关系。main 函数内容和 bubbleSort 函数的定义都不允许修改。

```
#include <iostream>
#include <cstring>
using namespace std;
template <class T>
void bubbleSort(T *array_start, T *array_end)
{
    // 参考教材 90 页的冒泡排序进行修改。
    for(T *pi = array_end; pi > array_start; pi--){        // 设定每轮冒泡的右边界，3 分；
        bool flag = true;                                  // 冒泡排序标记，1 分；
        for(T *pj = array_start; pj < pi; pj++){           // 设定该轮中交换的对象，3 分
            if(*pj > *(pj+1)){                             // 判断左右两个数的大小，2 分
                T tmp = *pj;                               // 交换左右两个数，4 分
                *pj = *(pj+1);
                *(pj+1) = tmp;

                flag = false;                              // 修改标记，1 分
            }
        }
        if(flag) break;                                    // 如果已经排好序，则退出，1 分
    }
}

int main(){
    int a[6] = {10, 6, 3, 9, 2, 7};
    bubbleSort(a, a+5);        // 模板函数对整数数组排序
    for(int i = 0; i <= 5; i++){
        cout << a[i] << " ";
    }
```

```
    cout << endl;

    double b[7] = {4.3, 6.5, 2.1, 3.4, 8.7, 3.8, 2.4};
    bubbleSort(b, b+6);        // 模板函数对浮点数数组排序
    for(int i = 0; i <= 6; i++){
        cout << b[i] << " ";
    }
    cout << endl;

    char c[8] = "fdsaodi";   // 模板函数对字符串数组排序
    bubbleSort(c, c+6);
    cout << c << endl;

    return 0;
}
```

5.（递归。 15 分）

递归编程：编写程序将字符串中的英文字母部分进行翻转输出。请用递归完成其中的函数：

```
    void reverseString(const char* str);

    例如：reverseString(“New”);
          输出：weN

          reverseString(“New123”);
          输出：123weN

          reverseString(“New123Year”);
          输出：123raeYweN
```

```
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;
void reverseString(const char* str);

/*
  本题中可能会使用的库函数说明如下：
  bool isalpha(char c)函数能判断一个字符是否为一个字母字符。
*/
int main()
{
    reverseString("New");   // 输出：weN
    cout << endl;
    reverseString("New123"); // 输出：123weN
    cout << endl;
    reverseString("New123Year");   //输出：123raeYweN
```

```
    cout << endl;
    return 0;
}

void reverseString(const char* str)
{
    if(str[0] == '\0')                  //   4 分
        return;
    if(isalpha(str[0])){                //   3 分
        reverseString(str+1);           //   2 分
        cout << str[0];                 //   2 分
    }
    else{
        cout << str[0];                 //   2 分
        reverseString(str+1);           //   2 分
    }
 }
```

6.（类。 20 分）

采用接口（.h 文件）与实现（.cpp 文件）分离的多文件方法，创建一个多项式类，完成其构造函数，重载赋值运算符=, >>, + 等运算符，在 main 主程序中测试你所编写的函数。本题头文件"polynomial.h"和"main.cpp"已经写好，无需改动。你只需要在类的实现"6.cpp" 文件中补充相应函数缺少的代码。

```
// 文件  main.cpp
#include <iostream>
#include "polynomial.h"
using namespace std;
int main()
{
      double dc[6]={1.0, 2.0, 3.0, 4.0, 0, 3.2};

      Polynomial p1(5,dc);   //调用构造函数
      Polynomial p2=p1;   //调用拷贝构造函数
      Polynomial p3;     // 调用默认构造函数
      Polynomial p4;

      cout << "p1 = " << p1 << endl;   // 调用重载运算符 <<
      cout << "p2 = " << p2 << endl;
      cout << "p3 = " << p3 << endl<<endl;

      p3 += p1;   //调用 += 函数
      cout << p3 << endl;

      cin >> p3;              // 调用重载运算符>>
      cout << "你输入的 p3 为: " << p3 << endl << endl;

      p4 = p1 + p3;       // 调用重载运算符 + 以及 赋值运算符 =
      cout << "p4 = p1 +p3" << endl;
```

```
    cout << "     = (" << p1 << ") + (" << p3 << ")" << endl << "     = " << p4    <<endl <<endl;

    return 0;
}

// 文件  Polynomial.h
#include <iostream>
using namespace std;

class Polynomial {
        int degree; //  多项式的次数，最高次项的次数。
        double *coefficients;
        //该指针指向的动态数组从第一个元素开始依次存储常数项，一次项系数，…,
        //最后元素是最高次项系数

        friend Polynomial operator +(const Polynomial & px, const Polynomial & py); //多项式相加
        friend ostream& operator << (ostream & os, const Polynomial & px);
        //  函数按照多项式的格式，从左到右依次打印高次项，…，常数项。
        //  例如打印四次多项式：    5x^4 + 23x^3 - x + 2  ；系数为 0 的项不打印；区别+，-符号。

        friend istream& operator >> (istream & os, Polynomial & px);

   public:
        Polynomial(int n,double *p); //带参数的构造函数
        Polynomial(); //默认的构造函数
        Polynomial(const Polynomial & px);   //  拷贝构造函数
        Polynomial& operator =(const Polynomial & px); //赋值操作符
        Polynomial& operator +=(const Polynomial & px); //+=操作符
        ~Polynomial();
};

//  类的实现文件  6.cpp
#include <iostream>
#include "polynomial.h"

Polynomial::Polynomial(int n, double *p)     //  要求数组 p 的元素个数等于 n
{
       degree =n;                          //----     1 分
       coefficients= new double[n+1];   // n+1 项, 2  分
       for(int i=0; i<=n;i++)
           coefficients[i] = p[i];       // 2 分

}

Polynomial::Polynomial()     //  默认构造函数
{
       degree =0;
       coefficients= new double[1];   // n+1 项
       coefficients[0] = 0;
}

Polynomial::Polynomial(const Polynomial & px)   //  拷贝构造函数
{
      degree = px.degree;
      coefficients= new double[px.degree+1];
```

```
    for(int i=0; i<=px.degree;i++)
        coefficients[i] = px.coefficients[i];
}


Polynomial& Polynomial::operator =(const Polynomial & px) //赋值操作符
{
    if(this==&px)
        return *this;       // 1 分
    delete [] coefficients;     // 1 分

    degree = px.degree;
    coefficients= new double[px.degree+1];      // 2 分
    for(int i=0; i<=px.degree;i++)
        coefficients[i] = px.coefficients[i];       // 1 分

    return *this;
}

Polynomial& Polynomial::operator +=(const Polynomial & px) //+=操作符
{
    *this = *this+px;
    return *this;
}

Polynomial operator +(const Polynomial & px, const Polynomial & py)
{
    int m=px.degree, n=py.degree;
    Polynomial tmp;
    int high = (m>=n) ? m:n;

    tmp.degree=high;
    tmp.coefficients = new double[high+1];

    int i;
    if(m>=n){           // 1 分
        for(i=0; i<=n; i++)
            tmp.coefficients[i] = px.coefficients[i]+py.coefficients[i];        // 1 分

            while(i<=m){        //该循环 1 分
                tmp.coefficients[i] = px.coefficients[i];
                i++;
             }
    }else                                   //  以下处理类似--- 2 分
        {
             for(int i=0; i<=m; i++)
                tmp.coefficients[i] = px.coefficients[i]+py.coefficients[i];
            while(i<=n){
                tmp.coefficients[i] = py.coefficients[i];
                i++;
            }
    }

    return tmp;
}
```

```
ostream& operator << (ostream & os, const Polynomial & px)
{
    int n=px.degree ;

    //处理除了一次项和常数项之外的输出
    for(int i=n ; i > 1; i--)
    {
        if(px.coefficients[i]>0)
        {
            if(i!=n)
                cout << " + ";
            cout <<px.coefficients[i];
            cout << "x^" << i;
        }
        else if(px.coefficients[i] < 0)
        {
            if(i!=n){
               cout << " - " << -px.coefficients[i];
            }
            else
                cout << px.coefficients[i];
            cout << "x^" << i;
        }
    }
    //单独处理一次项的输出
    if((px.coefficients[1] > 0) && (px.degree >=1))
    {
        cout << " + ";
        cout << px.coefficients[1] << "x";
    }
    else
        if(px.coefficients[1] < 0)
        {
            cout << px.coefficients[1] << "x";
        }
    //单独处理常数项的输出
    if(px.coefficients[0] >= 0)
        cout << " + ";
    cout << px.coefficients[0];

    return os;

}

istream& operator >> (istream & is, Polynomial & px)
{
    cout << "请输入多项式的次数:";
    int prevDegree = px.degree;     // 将多项式 px 原来的次数保存在临时变量中
    is >> px.degree;     // 输入多项式的次数， 注意：新输入的次数可能要大于之前的次数

    //请完成此处缺少的代码  (5 分)  提示：注意如果 px.degree > prevDegree 需要做什么处理？
    if(px.degree > prevDegree)     // 1 分
    {
        delete [] px.coefficients;       // 2 分
        px.coefficients = new double[px.degree+1];   //2 分
    }
```

```
    cout << "请依次从常数项到高次项，输入多项式的系数 (系数用空格隔开，中间有系数为 0 时不得省略):" << endl;

    for(int i=0;i<= px.degree;i++)
        is >> px.coefficients[i];

    return is;
}


Polynomial::~Polynomial()
{
    delete [] coefficients;

}
```