## 设计一个函数，判别一个整数是否为素数。

#include <iostream>

using namespace std;

bool isPrime(int n) {

if (n == 1)

return false;

if (n == 2)

return true;

int limit = sqrt(n) + 1;

for (int i = 2; i < limit; i++)

if (n % i == 0)

return false;

return true;

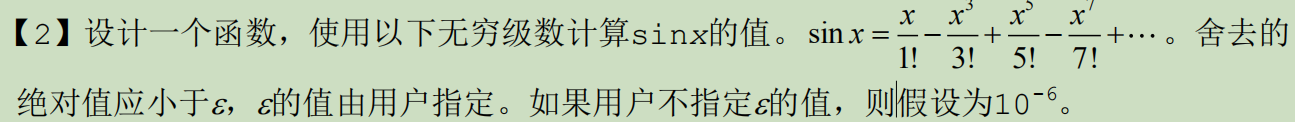
}

int main() {

cout << isPrime(4) << endl;

}

## 设计一个函数，使用以下无穷级数计算sin*x*的值



#include <iostream>

#include <iomanip>

#define PI 3.1415926

using namespace std;

double Sin(double x, double epsilon = 1e-6) {

double sin = 0., item = x;

for (int i = 2; fabs(item) > epsilon; i++) {

sin += item;

item = -1 \* item \* x \* x / (2 \* i - 1) / (2 \* i - 2);

}

return sin;

}

int main() {

cout << fixed << setprecision(5) << Sin(PI) << endl;

}

## 写一个函数stringCopy将一个字符串复制到另一个字符串。

#include <iostream>

using namespace std;

void stringCopy(char src[], char des[]) {

int i = 0;

for ( i = 0; src[i] != '\0'; i++)

des[i] = src[i];

des[i] = '\0';

}

int main() {

char src[] = "abcd";

char des[5];

stringCopy(src, des);

for (int i = 0; i < 5; i++)

cout << des[i];

}

## 设计一个支持整型、实型和字符型的气泡排序的函数模板

#include <iostream>

using namespace std;

template <class T>

void bubbleSort(T data[], int size) {

for (int i = 0; i < size - 1; i++) {

bool flag = false;

for (int j = size - 1; j > i; j--) {

if (data[j] < data[j - 1]) {

T temp = data[j];

data[j] = data[j - 1];

data[j - 1] = temp;

}

}

if (flag == true)//本次比较，没有交换

return;

}

}

int main() {

char str[5] = "adcb";

bubbleSort(str, 4);

for (int i = 0; i < 5; i++)

cout << str[i];

}

## 设计一函数求两个正整数的最大公约数。

#include <iostream>

using namespace std;

int gcb(int a,int b) {

for (int i = (a > b ? b : a); i > 0; i--)

if (a % i == 0 && b % i == 0)

return i;

}

int main() {

cout << gcb(4, 2);

}

## 设计一个用于整型数的二分查找的递归函数。

#include <iostream>

using namespace std;

int binarySearch(int data[], int start, int end, int x) {

int mid = (start + end) / 2;

if (start > end) return -1;

if (data[mid] == x) return mid;

if (data[mid] > x)

return binarySearch(data, start, mid - 1, x);

else

return binarySearch(data, mid + 1, end, x);

}

int main() {

int data[]{ 1,2,3,4,5 };

cout << binarySearch(data, 0, 4, 2);

}

## 设计一个支持整型、实型和字符型数据的快速排序的函数模板。

## 设计一函数，用动态规划求Fibonacci(n)。

int Fibonacci(int n) {

const int max = 100;

int f[max];

if (n == 0)

return 0;

if (n == 1)

return 1;

for (int i = 2; i <= n; i++)

f[i] = f[i - 1] + f[i - 2];

return f[n];

}

## 设计一个函数，输出小于100的所有的Fibonacci数。

void Fibonacci() {

double f[101];

f[0] = 1;

f[1] = 1;

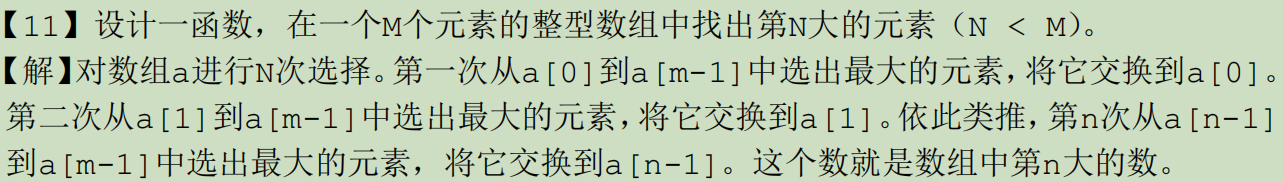
cout << 1 << "\t" << 1 << "\t";

for (int i = 2; i <= 100; i++)

cout << (f[i] = f[i - 1] + f[i - 2]) << "\t";

}

## 设计一函数，在一个M个元素的整型数组中找出第N大的元素（N < M）。



int findMaxN(int data[], int len, int n) {

int max, index = 0;

for (int i = 0; i < n; i++) {

index = i; max = data[i];

for (int j = i; j < len; j++) {

if (data[j] > max) {

index = j; max = data[j];

}

}

data[index] = data[i];

data[i] = max;

}

return max;

}

## 编写一函数int count()，使得第一次调用时返回1，第二次调用时返回2。即返回当前的调用次数。

int Count() {

static int count = 0;

return ++count;

}

## 12. 用结构体表示一个复数，编写实现复数的加法、乘法、输入和输出的函数，并测试这些函数。

#include <iostream>

using namespace std;

struct complex

{

int real;

int imag;

};

void input(complex& a) {

cin >> a.real >> a.imag;

}

complex Add(complex a,complex b) {

complex result;

result.real = a.real + b.real;

result.imag = a.imag + b.imag;

return result;

}

complex mutiple(complex a, complex b) {

complex result;

result.real = a.real \* b.real - a.imag \* b.imag;

result.imag = a.real \* b.imag + b.real \* a.imag;

return result;

}

void Print(complex a) {

cout << a.real << '+' << a.imag << 'i' << endl;

}

int main() {

complex A;

input(A);

Print(A);

}

## 13. 定义表示有理数的结构体类型。它有两个整型的成员，分别表示分子和分母。

#include <iostream>

using namespace std;

struct rational

{

int num;//分子

int den;//分母

};

rational CreateRational(int num, int den) {

rational r;

r.den = den;

r.num = num;

//化简

int fac = (abs(r.den) < abs(r.num) ? abs(r.den) : abs(r.num));//源代码有bug

while (r.den % fac != 0 || r.num % fac != 0) fac--;

r.den /= fac;

r.num /= fac;

return r;

}

rational AddRational(rational r1, rational r2) {

rational r;

r.num = r1.num \* r2.den + r1.den \* r2.num;

r.den = r1.den \* r2.den;

//化简

int fac = (abs(r.den) < abs(r.num) ? abs(r.den) : abs(r.num));//源代码有bug

while (r.den % fac != 0 || r.num % fac != 0) fac--;

r.den /= fac;

r.num /= fac;

return r;

}

rational MultiplyRational(rational r1, rational r2) {

rational r;

r.den = r1.den \* r2.den;

r.num = r1.num \* r2.num;

//化简

int fac = (abs(r.den) < abs(r.num) ? abs(r.den) : abs(r.num));//源代码有bug

while (r.den % fac != 0 || r.num % fac != 0) fac--;

r.den /= fac;

r.num /= fac;

return r;

}

double GetRational(rational r) {

return (double)r.num / r.den;

}

void PrintRational(rational r) {

cout << r.num << '/' << r.den << endl;

}

int main() {

rational r1 = CreateRational(-3, 4);

PrintRational(r1);

}

## 14. 设计一个字符串处理库，该库提供一组常用的字符串的操作，包括字符串复制、字符 串拼接、字符串比较、求字符串长度和取字符串的子串。

void stringCopy(char\* des, const char\* src) {

int i = 0;

for (; src[i] != '\0'; i++)

des[i] = src[i];

des[i] = '\0';

}

void stringNCopy(char\* des, const char\* src, int len) {

int i = 0;

for (; src[i] != '\0' && i < len; i++)

des[i] = src[i];

des[i] = '\0';

}

void stringCat(char\* des, const char\* src) {

int i = 0;

for (; des[i] != '\0'; i++);

int j = 0;

for (; src[j] != '\0'; i++, j++)

des[i] = src[j];

des[i] = '\0';

}

void stringNCat(char\* des, const char\* src, int len) {

int i = 0;

for (; des[i] != '\0'; i++);

int j = 0;

for (; src[j] != '\0' && j < len; i++, j++)

des[i] = src[j];

des[i] = '\0';

}

bool stringCmp(const char\* s1, const char\* s2) {

int i = 0, j = 0;

for (; s1[i] != '\0' && s2[j] != '\0'; i++, j++)

if (s1[i] != s2[j])

return false;

if (s1[i] != s2[j])

return false;

return true;

}

int stringLen(const char\* s) {

int i=0;

for (; s[i] != '\0'; i++);

return i;

}

int main() {

char\* des = new char[10]{ "efgh" };

char\* src = new char[5]{ "abcd" };

stringNCat(des, src, 1);

//stringNCopy(des, src,2);

for (int i = 0; des[i] != '\0'; i++)

cout << des[i];

}

## 15. 创建一个处理任意大的正整数的类 LongLongInt，用一个动态的字符数组存放任意 长度的正整数。数组的每个元素存放整型数的一位。例如，123 被表示为“321”。注意， 数字是逆序存放，这样可以使得整型数的操作比较容易实现。提供的成员函数有构造函数（根 据一个由数字组成的字符串创建一个 LongLongInt 类的对象）、输出函数、加法函数、 把一个 LongLongInt 类的对象赋给另一个对象的赋值函数。为了比较 LongLongInt 对象，提供了等于比较、大于比较、大于等于比较。

#pragma once

#include <iostream>

using namespace std;

class LongLongInt

{

private:

char\* num;

public:

LongLongInt(const char\* n = "");//构造函数

LongLongInt(const LongLongInt&);//赋值构造函数

~LongLongInt() { delete num; }

void Print()const;

void Add(const LongLongInt&, const LongLongInt&);

void assign(const LongLongInt&);

bool equal(const LongLongInt&)const;

bool greater(const LongLongInt&)const;

bool greaterOrEqual(const LongLongInt&)const;

};

#include "LongLongInt.h"

LongLongInt::LongLongInt(const char\* n = "")//构造函数

{

int len = strlen(n);//注意此处

this->num = new char[len + 1];//+1是'\0'

for (int i = 0; n[i] != '\0'; i++)

this->num[len - i - 1] = n[i];

num[len] = '\0';

}

LongLongInt::LongLongInt(const LongLongInt& other)//赋值构造函数

{

this->num = new char[strlen(other.num) + 1];

strcpy(this->num, other.num);

}

void LongLongInt::Print()const {

for (int i = strlen(this->num) - 1; i > -1; i--)

cout << this->num[i];

if (strlen(num) == 0)

cout << 0;

}

void LongLongInt::Add(const LongLongInt& n1, const LongLongInt& n2) {

int len1 = strlen(n1.num), len2 = strlen(n2.num);

int minLen = (len1 > len2 ? len2 : len1);

int len = (len1 > len2 ? len1 : len2) + 1;

int carry = 0, result; // carry：进位

num = new char[len + 1];

for (int i = 0; i < minLen; ++i) { // n1和n2都有数字

result = n1.num[i] - '0' + n2.num[i] - '0' + carry;

num[i] = result % 10 + '0';

carry = result / 10;

}

while (i < len1) { // n2已结束

result = n1.num[i] - '0' + carry;

num[i] = result % 10 + '0';

carry = result / 10;

++i;

}

while (i < len2) { // n1已结束

result = n2.num[i] - '0' + carry;

num[i] = result % 10 + '0';

carry = result / 10;

++i;

}

if (carry != 0) num[i++] = carry + '0'; // 处理最高位的进位

num[i] = '\0';

if (i != len) { // 最高位无进位处理

char\* tmp = num;

num = new char[len];

strcpy(num, tmp);

delete tmp;

}

}

void LongLongInt::assign(const LongLongInt& other) {

if (this == &other) return;//数首先检查right是否就是当前对象

delete this->num;

num = new char[strlen(other.num) + 1];

strcpy(this->num, other.num);

}

bool LongLongInt::equal(const LongLongInt& other)const {

return strcmp(this->num, other.num) == 0;

}

bool LongLongInt::greater(const LongLongInt& other)const {

int len1 = strlen(this->num), len2 = strlen(other.num);

if (len1 > len2) return true;

if (len1 < len2) return false;

for (int i = len1 - 1; i > -1; i--) {

if (num[i] > other.num[i]) return true;

if (num[i] < other.num[i]) return false;

}

return false;//相等

}

bool LongLongInt::greaterOrEqual(const LongLongInt& other)const {

int len1 = strlen(this->num), len2 = strlen(other.num);

if (len1 > len2) return true;

if (len1 < len2) return false;

for (int i = len1 - 1; i > -1; i--) {

if (num[i] > other.num[i]) return true;

if (num[i] < other.num[i]) return false;

}

return true;

}

## 16. 完善本章提到的SavingAccount类。该类的属性有账号、存款金额和月利率。账号自 动生成。第一个生成的对象账号为1，第二个生成的对象账号为2，依次类推。所需的操作 有修改利率、每月计算新的存款额（原存款额+本月利息）和显示账户金额。

#include <iostream>

using namespace std;

class SavingAccount

{

private:

int no;

double balance;

static double rate;

static int totalNo;

public:

SavingAccount(double deposit);

void updateMonthly();

void print() const;

static void setRate(double);//静态函数才能对静态变量操作

static int generateNo();

};

#include "SavingAccount.h"

double SavingAccount::rate = 0;//static初始化在类外

int SavingAccount::totalNo = 0;

SavingAccount::SavingAccount(double deposit) {

this->balance = deposit;

this->no = generateNo();

}

void SavingAccount::updateMonthly() {

this->balance += this->balance \* this->rate;

}

void SavingAccount::print() const {

cout << this->no << "\t" << balance << endl;

}

void SavingAccount::setRate(double newRate) {//此处不能写static

rate = newRate;

}

int SavingAccount::generateNo() {

return totalNo++;

}

## 17. 试定义一个string类，用以处理字符串。它至少具有两个数据成员：字符串的内容和长度。提供的操作有显示字符串、求字符串长度、在原字符串后添加一个字符串等（不能用cstring库）。

#pragma once

#include <iostream>

using namespace std;

class String

{

private:

int len;

char\* data;

public:

String(const char\* s = "");

String(const String& s);

void Copy(const String& src);

void Cat(const String& src);

int Len() const { return len; };

void Print() const { cout << data; }

~String() { delete data; }

};

#include "String.h"

String::String(const char\* s) {//不能写默认参数

this->len = strlen(s);

this->data = new char[this->len + 1];

for (int i = 0; s[i] != '\0'; i++)

this->data[i] = s[i];

this->data[this->len] = '\0';

}

String::String(const String& s) {

this->len = s.len;

this->data = new char[this->len + 1];

for (int i = 0; s.data[i] != '\0'; i++)

this->data[i] = s.data[i];

this->data[this->len] = '\0';

}

void String::Copy(const String& src) {

delete this->data;//释放之前的空间

this->len = src.len;

this->data = new char[this->len + 1];

for (int i = 0; src.data[i] != '\0'; i++)

this->data[i] = src.data[i];

this->data[this->len] = '\0';

}

void String::Cat(const String& src) {

char\* tmp = this->data;

int i;

data = new char[src.len + this->len + 1];

for (i = 0; tmp[i] != '\0'; i++)

this->data[i] = tmp[i];

for (i = 0; src.data[i] != '\0'; i++)

this->data[len + i] = src.data[i];

this->len += src.len;

this->data[len] = '\0';

}

## 18. 为学校的教师提供一个工具，使教师可以管理自己所教班级的信息。教师所需了解和 处理的信息包括课程名、上课时间、上课地点、学生名单、学生人数、期中考试成绩、期末考试成绩和平时的课堂练习成绩。每位教师可自行规定课堂练习次数的上限。考试结束后， 该工具可为教师提供成绩分析，统计最高分、最低分、平均分及优、良、中、差的人数。

#include <iostream>

using namespace std;

class Teacher

{

private:

char className[30];

char time[20];

char place[20];

int numOfStudent;//学生人数

int totalQuiz;//总的小测验数

int curQuiz;//已进行的小测验数

struct student {

char name[10];

int scoreMid;//期中成绩

int scoreFinal;//期末成绩

int\* scoreQuiz;//保存每次小测验成绩

};

student\* sInfo;

public:

enum Type { MID, FINAL, QUIZ };

Teacher(char\* cName, char\* cTime, char\* cPlace, int noS, int noQuiz);

~Teacher();

void inputStudent();

void inputScore(Type);

void analysis(Type);

};

#include "Teacher.h"

//初始化列表赋值

Teacher::Teacher(char\* cName, char\* cTime, char\* cPlace, int noS, int noQuiz) :numOfStudent(noS), totalQuiz(noQuiz)

{

strcpy(this->className, cName);//char要记得str函数！！！

strcpy(this->time, cTime);

strcpy(this->place, cPlace);

sInfo = new student[this->numOfStudent];

for (int i = 0; i < this->numOfStudent; i++)

sInfo[i].scoreQuiz = new int[totalQuiz];

}

Teacher::~Teacher() {//注意析构函数

for (int i = 0; i < this->numOfStudent; i++)

delete sInfo[i].scoreQuiz;

delete sInfo;

}

void Teacher::inputStudent() {

for (int i = 0; i < this->numOfStudent; i++) {

cout << "请输入第" << i << "个同学姓名：";

cin >> sInfo[i].name;

}

}

void Teacher::inputScore(Type scoretype) {

switch (scoretype)

{

case Teacher::MID:

for (int i = 0; i < numOfStudent; i++) {

cout << "请输入" << sInfo[i].name[i] << "的期中成绩：";

cin >> sInfo[i].scoreMid;

}

break;

case Teacher::FINAL:

for (int i = 0; i < numOfStudent; i++) {

cout << "请输入" << sInfo[i].name[i] << "的期末成绩：";

cin >> sInfo[i].scoreFinal;

}

break;

case Teacher::QUIZ:

if (curQuiz > totalQuiz) {

cout << "所有测验都已完成！无法输入。" << endl;

return;

}

for (int i = 0; i < numOfStudent; ++i) {

cout << "请输入" << sInfo[i].name << "的测验成绩：";

cin >> sInfo[i].scoreQuiz[curQuiz];

}

++curQuiz;

break;

default:

break;

}

}

void Teacher::analysis(Type) {

懒得写了

}