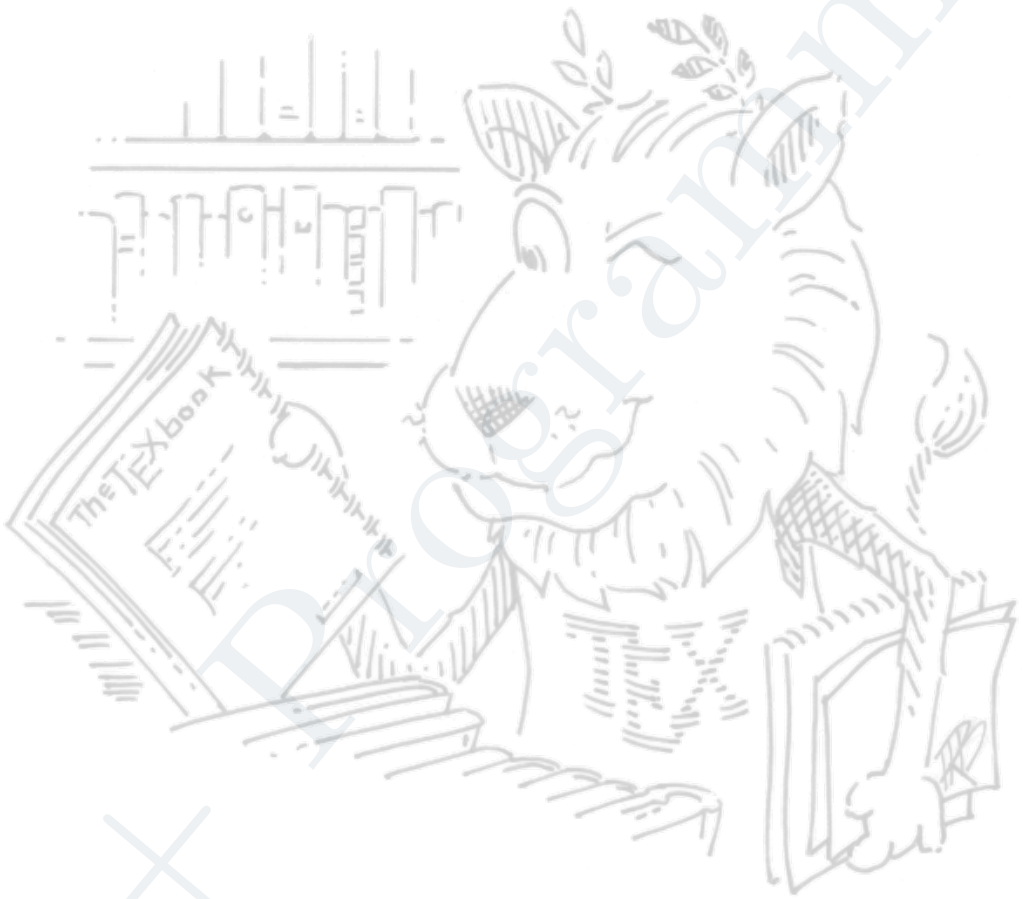
《C++ 程序设计》OJ3（参考答案）

VCG

2023 年 5 月 27 日



1 Singer 类

【描述】

实现一个 Singer 类，通过以下测试：

```
int main()
{
    Singer s1, s2;
    cin >> s1 >> s2;
    cout << s1 << "\n" << s2 << endl;

    if (s1 > s2)
        cout << s1.getName() << "'s score is higher than " << s2.getName() << "'s.\
            n";
    else if (s1 == s2)
        cout << s1.getName() << "'s score is equal to " << s2.getName() << "'s.\n";
    else
        cout << s1.getName() << "'s score is lower than " << s2.getName() << "'s.\n
            ";

    return 0;
}
```

【输入描述】

输入包含两行。
第一行为歌手 s1 的信息。第二行为歌手 s2 的信息，每位歌手的信息包括姓名（不包含空格）、性别、年龄和分数；姓名、性别、年龄和分数之间用空格分隔

【输出描述】

输出为三行。
前两行分别是歌手 s1 和 s2 的信息。第三行根据 s1 和 s2 比较结果输出，具体参见主函数

【输入样例】

```
Mary F 28 99.5
Peter M 26 98
```

【输出样例】

```
Mary F 28 99.5
Peter M 26 98
Mary's score is higher than Peter's.
```

【提示】

main.cpp

```cpp
#include <iostream>
#include <string>

class Singer
{
private:
    std::string name;
    char gender;
    int age;
    double score;
public:
    Singer(std::string name = "", char gender = '?', int age = 0, double score = 0.0);
    std::string getName();
    friend bool operator>(const Singer&, const Singer&);
    friend bool operator<(const Singer&, const Singer&);
    friend bool operator==(const Singer&, const Singer&);
    friend std::istream& operator>>(std::istream&, Singer&);
    friend std::ostream& operator<<(std::ostream&, const Singer&);
};

Singer::Singer(std::string name, char gender, int age, double score)
{
    this->name = name;
    this->gender = gender;
    this->age = age;
    this->score = score;
}

std::string Singer::getName()
{
    return name;
}

bool operator>(const Singer& s1, const Singer& s2)
{
    return s1.score > s2.score;
}

bool operator<(const Singer& s1, const Singer& s2)
{
    return s1.score < s2.score;
}

bool operator==(const Singer& s1, const Singer& s2)
{
    return s1.score == s2.score;
}
```

```
48
49  std::istream& operator>>(std::istream& is, Singer& s)
50  {
51      std::cin >> s.name >> s.gender >> s.age >> s.score;
52      return is;
53  }
54
55  std::ostream& operator<<(std::ostream& os, const Singer& s)
56  {
57      std::cout << s.name << " " << s.gender << " " << s.age << " " << s.score;
58      return os;
59  }
60
61  int main()
62  {
63      Singer s1, s2;
64      std::cin >> s1 >> s2;
65      std::cout << s1 << "\n" << s2 << std::endl;
66
67      if (s1 > s2)
68          std::cout << s1.getName() << "'s score is higher than " << s2.getName() <<
                "'s.\n";
69      else if (s1 == s2)
70          std::cout << s1.getName() << "'s score is equal to " << s2.getName() << "'s
                .\n";
71      else
72          std::cout << s1.getName() << "'s score is lower than " << s2.getName() << "
                's.\n";
73
74      return 0;
75  }
```

2 Sales_data 类

【描述】

实现 Sales_data 类（包括它的友元函数）：

```
class Sales_data
{
    //输入书号、销量和收入
    friend istream & operator>>(istream&, Sales_data &);
    //输出书号、销量、收入和均价
    friend ostream & operator<<(ostream &, const Sales_data &);
    friend bool operator==(const Sales_data &, const Sales_data &);
    friend bool operator!=(const Sales_data &, const Sales_data &);
    // for "+", assume that both objects refer to the same book
    friend Sales_data operator+(const Sales_data &, const Sales_data &);

public:
    Sales_data() : units_sold(0), revenue(0.0) {}
    Sales_data(const string & s, unsigned n, double r) :
        bookNo(s), units_sold(n), revenue(r)
    {}
    string get_bookNo() const;
    // for "+=", assume that both objects refer to the same book
    Sales_data & operator+=(const Sales_data &);

private:
    double avg_price() const;  //均价，等于收入除以销量
    string bookNo;          //书号
    unsigned units_sold; //销量
    double revenue;         //收入
};
```

通过以下 main 函数的测试

```
int main()
{
    Sales_data item1, item2;
    while(std::cin >> item1 >> item2)
    {
        std::cout << item1 << "\n" << item2 << "\n";
        if (item1 == item2){
            std::cout << item1.get_bookNo() << " equals " << item2.get_bookNo() <<
                "\n";
            std::cout << (item1 + item2) << "\n";
            item1 += item2;
            std::cout << item1 << "\n";
        }
        if (item1 != item2)
            std::cout << item1.get_bookNo() << " doesn't equal " << item2.
                get_bookNo() << "\n";
    }
```

```
16      return 0;
17  }
```

【输入描述】

输入多组数据，每组数据两行，每行表示 1 个 Sale_data 对象，依次是书号、销量和收入

【输出描述】

对于每组数据，输出具体参见 main 函数和输出样例

【输入样例 1】

```
001 10 100.0
001 10 100.0
```

【输出样例 1】

```
001 10 100 10
001 10 100 10
001 equals 001
001 20 200 10
001 20 200 10
```

【输入样例 2】

```
002 5 250
003 8 400
```

【输出样例 2】

```
002 5 250 50
003 8 400 50
002 doesn't equal 003
```

【提示】

main.cpp

```cpp
#include<string>
#include<iostream>

class Sales_data
{
    //输入书号、销量和收入
    friend std::istream& operator>>(std::istream&, Sales_data&);
    //输出书号、销量、收入和均价
    friend std::ostream& operator<<(std::ostream &, const Sales_data&);
    friend bool operator==(const Sales_data&, const Sales_data&);
    friend bool operator!=(const Sales_data&, const Sales_data&);
    // for "+", assume that both objects refer to the same book
    friend Sales_data operator+(const Sales_data&, const Sales_data&);

public:
    Sales_data() : units_sold(0), revenue(0.0) {}
    Sales_data(const std::string& s, unsigned n, double r) :
        bookNo(s), units_sold(n), revenue(r){}
    std::string get_bookNo() const;
    // for "+=", assume that both objects refer to the same book
    Sales_data & operator+=(const Sales_data&);

private:
    double avg_price() const; //均价，等于收入除以销量
    std::string bookNo;       //书号
    unsigned units_sold; //销量
    double revenue;      //收入
};

std::string Sales_data::get_bookNo() const { return bookNo; }

double Sales_data::avg_price() const
{
    if (units_sold)
        return revenue / units_sold;
    else
        return 0;
}

Sales_data& Sales_data::operator+=(const Sales_data& rhs)
{
    units_sold += rhs.units_sold;
    revenue += rhs.revenue;
    return *this;
}

Sales_data operator+(const Sales_data& lhs, const Sales_data& rhs)
{
```

```
49      Sales_data sum = lhs;  // copy data members from lhs into sum
50      sum += rhs;            // add rhs into sum
51      return sum;
52  }
53
54  std::istream& operator>>(std::istream & is, Sales_data& item)
55  {
56      is >> item.bookNo >> item.units_sold >> item.revenue;
57      return is;
58  }
59
60  std::ostream& operator<<(std::ostream& os, const Sales_data& item)
61  {
62      os << item.get_bookNo() << " " << item.units_sold << " "
63          << item.revenue << " " << item.avg_price();
64      return os;
65  }
66
67  bool operator==(const Sales_data& lhs, const Sales_data& rhs)
68  {
69      return lhs.get_bookNo() == rhs.get_bookNo() &&
70          lhs.units_sold == rhs.units_sold &&
71          lhs.revenue == rhs.revenue;
72  }
73
74  bool operator!=(const Sales_data& lhs, const Sales_data& rhs)
75  {
76      return !(lhs == rhs);
77  }
78
79  int main()
80  {
81      Sales_data item1, item2;
82      while(std::cin >> item1 >> item2)
83      {
84          std::cout << item1 << "\n" << item2 << "\n";
85          if (item1 == item2){
86              std::cout << item1.get_bookNo() << " equals " << item2.get_bookNo() <<
                    "\n";
87              std::cout << (item1 + item2) << "\n";
88              item1 += item2;
89              std::cout << item1 << "\n";
90          }
91          else if (item1 != item2)
92              std::cout << item1.get_bookNo() << " doesn't equal " << item2.
                    get_bookNo() << "\n";
93      }
94      return 0;
95  }
```

3 String 类

【描述】

实现 String 类并使用以下的 main 函数进行测试：

```
class String{
private:
    char * s;
public:
    String();
    String(const char *);
    String(const String &);
    ~String();
    String & operator=(const char *);
    String & operator=(const String &);
    String operator+(const char *);
    String operator+(const String &);
    String & operator+=(const char *);
    String & operator+=(const String &);
    friend istream & operator>>(istream &, String &);
    friend ostream & operator<<(ostream &, const String &);
    friend bool operator==(const String &, const char *);
    friend bool operator==(const String &, const String &);
    friend bool operator!=(const String &, const char *);
    friend bool operator!=(const String &, const String &);
};

int main()
{
    String s;
    s += "hello";
    cout << s << endl;
    String s1("String1");
    String s2("copy of ");
    s2 += "String1";
    cout << s1 << "\n" << s2 << endl;
    String s3;
    cin >> s3;
    cout << s3 << endl;
    String s4("String4"), s5(s4);
    cout << (s5 == s4) << endl;
    cout << (s5 != s4) << endl;
    String s6("End of "), s7("my string.");
    s6 += s7;
    cout << s6 << endl;
    return 0;
}
```

【输入描述】

s3 的值

【输出描述】

一连串 String 类的字符串，详见样例

【输入样例】

```
String3
```

【输出样例】

```
hello
String1
copy of String1
String3
1
0
End of my string.
```

【提示】

main.cpp

```cpp
#define _CRT_SECURE_NO_WARNINGS

#include<iostream>
#include<cstring>
#include<cstdlib>

class String{
private:
    char* s;
public:
    String();
    String(const char*);
    String(const String&);
    ~String();
    String& operator=(const char*);
    String& operator=(const String&);
    String operator+(const char*);
    String operator+(const String&);
    String& operator+=(const char*);
    String& operator+=(const String&);
    friend std::istream& operator>>(std::istream&, String&);
    friend std::ostream& operator<<(std::ostream&, const String&);
    friend bool operator==(const String&, const char*);
    friend bool operator==(const String&, const String&);
    friend bool operator!=(const String&, const char*);
    friend bool operator!=(const String&, const String&);
};

String::String()
{
    s = new char[1];
    s[0] = '\0';
}

String::String(const char* sp)
{
    s = new char[strlen(sp) + 1];
    strcpy(s, sp);
}

String::String(const String& str)
{
    s = new char[strlen(str.s) + 1];
    strcpy(s, str.s);
}

String::~String(){ delete[] s; }

```

```
49 String& String::operator=(const char* sp)
50 {
51     delete[] s;
52     s = new char[strlen(sp) + 1];
53     strcpy(s, sp);
54     return *this;
55 }
56
57 String& String::operator=(const String& str)
58 {
59     if(this != &str)
60         *this = str.s;
61     return *this;
62 }
63
64 String String::operator+(const char* sp)
65 {
66     String temp;
67     temp.s = new char[strlen(s) + strlen(sp) + 1];
68     strcpy(temp.s, s);
69     strcat(temp.s, sp);
70     return temp;
71 }
72
73 String String::operator+(const String& str)
74 {
75     String temp;
76     temp = *this + str.s;
77     return temp;
78 }
79
80 String& String::operator+=(const char* sp)
81 {
82     return *this = *this + sp;
83 }
84
85 String& String::operator+=(const String& str)
86 {
87     return *this = *this + str;
88 }
89
90
91 std::istream& operator>>(std::istream& is, String& str)
92 {
93     char temp[255];
94     is >> temp;
95     str = temp;
96     return  is;
97 }
98
```

```cpp
std::ostream& operator<<(std::ostream& os, const String& str)
{
    os << str.s;
    return os;
}

bool operator==(const String& str, const char* sp)
{
    return strcmp(str.s, sp) == 0;
}

bool operator==(const String& str1, const String& str2)
{
    return strcmp(str1.s, str2.s) == 0;
}

bool operator!=(const String& str, const char* sp)
{
    return strcmp(str.s, sp) != 0;
}

bool operator!=(const String& str1, const String& str2)
{
    return strcmp(str1.s, str2.s) != 0;
}

int main()
{
    String s;
    s += "hello";
    std::cout << s << std::endl;
    String s1("String1");
    String s2("copy of ");
    s2 += "String1";
    std::cout << s1 << "\n" << s2 << std::endl;
    String s3;
    std::cin >> s3;
    std::cout << s3 << std::endl;
    String s4("String4"), s5(s4);
    std::cout << (s5 == s4) << std::endl;
    std::cout << (s5 != s4) << std::endl;
    String s6("End of "), s7("my string.");
    s6 += s7;
    std::cout << s6 << std::endl;

    return 0;
}
```

4 迭代器

【描述】

自增 (++) 和自减 (–) 操作符经常由诸如迭代器这样的类实现，这样的类提供类似于指针的行为访问序列中的元素。例如，可以定义一个类，该类指向一个数组并为该数组中的元素提供访问检查。假设有以下类，它将处理 int 数组。实现这个类并使用 main 函数测试此类。

```
class CheckedPtr{
public:
    CheckedPtr(int * b, int * e) : begin(b), end(e), current(b){
    }
    CheckedPtr & operator ++(); // prefix ++
    CheckedPtr & operator --(); // prefix --
    CheckedPtr   operator ++(int); // postfix ++
    CheckedPtr   operator --(int); // postfix --
    int* Begin();
    int* End();
    int* Current();
private:
    int *begin;     //pointer to beginning of the array
    int *end;       //one past the end of the array
    int *current;   //current position within the array
};

int main()
{
    int n;
    cin >> n;
    int * array = new int[n];

    for (int i = 0; i < n; i++)
        cin >> array[i];

    CheckedPtr cp(array, array+n);
    for(; cp.Current()<cp.End(); cp++)
        cout<<*cp.Current()<<" ";
    cout<<endl;

    for(--cp; cp.Current()>cp.Begin(); cp--)
        cout<<*cp.Current()<<" ";
    cout<<*cp.Current()<<endl;

    delete[] array;

    return 0;
}
```

【输入描述】

正整数 n
n 个整数

【输出描述】

上述 n 个整数的正序
上述 n 个整数的逆序

【输入样例】

```
5
1 2 3 4 5
```

【输出样例】

```
1 2 3 4 5
5 4 3 2 1
```

【提示】

main.cpp

```cpp
#include <iostream>

class CheckedPtr{
public:
    CheckedPtr(int* b, int* e): begin(b), end(e), current(b){}
    CheckedPtr& operator ++(); // prefix ++
    CheckedPtr& operator --(); // prefix --
    CheckedPtr operator ++(int); // postfix ++
    CheckedPtr operator --(int); // postfix --
    int* Begin();
    int* End();
    int* Current();
private:
    int *begin;         //pointer to beginning of the array
    int *end;         //one past the end of the array
    int *current;     //current position within the array
};

CheckedPtr& CheckedPtr::operator++(){
    if(current == end)
        std::cout << "increment past the end of CheckedPtr." << std::endl;
    else
        ++current;
    return *this;
}

CheckedPtr& CheckedPtr::operator--(){
    if(current == begin)
        std::cout << "decrement past the beginning of CheckedPtr." << std::endl;
    else
        --current;
    return *this;
}

CheckedPtr CheckedPtr::operator++(int){
    CheckedPtr ret(*this);
    ++*this;      //调用prefix ++
    return ret;
}

CheckedPtr CheckedPtr::operator--(int){
    CheckedPtr ret(*this);
    --*this;      //调用prefix --
    return ret;
}

int* CheckedPtr::Begin(){
    return begin;
```

```
49  }
50
51  int* CheckedPtr::End(){
52      return end;
53  }
54
55  int* CheckedPtr::Current(){
56      return current;
57  }
58
59  int main()
60  {
61      int n;
62      std::cin >> n;
63      int* array = new int[n];
64
65      for(int i = 0; i < n; i++)
66          std::cin >> array[i];
67
68      CheckedPtr cp(array, array+n);
69      for(;cp.Current()<cp.End(); cp++)
70          std::cout << *cp.Current() << " ";
71      std::cout << std::endl;
72
73      for(--cp; cp.Current()>cp.Begin(); cp--)
74          std::cout << *cp.Current() << " ";
75      std::cout << *cp.Current() << std::endl;
76
77      delete[] array;
78      return 0;
79  }
```

5 大整数

【描述】

编写高精度运算类 Bign，完成大整数的运算，类的数据成员有两个，一个是 int 型数据用来存储大数数据的有效长度，另一个是整型数组用来存储大整数。
要求支持字符串赋值和整数赋值，重载 + - * / » « += > < >= <= == != 等运算符。使用如下 main 函数测试你的程序

```
int main()
{
    UnsignedBigInt a, b, c;
    a = "12349987654321999999";
    c = 123456;
    cout << c << endl;
    while(cin >> b)
    {
        cout << a + b << endl;
        cout << a - b << endl;
        cout << (a < b) << endl;
        a += b;
        cout << a << endl;
    }
    return 0;
}
```

【输入描述】

一个大整数

【输出描述】

用以上 main 函数输出的结果

【输入样例】

```
651467465456418888888454524888
```

【输出样例】

```
123456
651467465579918765427765248887
-651467464222918901230225248889
1
651467465579918765427765248887
```

【提示】

main.cpp

```
#include<iostream>
#include<string>
#include<vector>

class UnsignedBigInt
{
    friend std::istream& operator>>(std::istream& is, UnsignedBigInt& bigint);
    friend std::ostream& operator<<(std::ostream& os, const UnsignedBigInt& bigint)
        ;
    friend UnsignedBigInt ADD(const UnsignedBigInt& a, const UnsignedBigInt& b)
    {
        unsigned int n = a.bits.size() > b.bits.size() ? a.bits.size() : b.bits.
            size();
        UnsignedBigInt result;
        result.bits.clear();
        int c = 0;
        for(unsigned int i = 0; i < n; ++i)
        {
            int sum = c;
            if(i < a.bits.size()) sum += a.bits[i];
            if(i < b.bits.size()) sum += b.bits[i];
            c = sum / 10;
            result.bits.push_back(sum % 10);
        }
        if(c)
            result.bits.push_back(c);
        return result;
    }
    friend UnsignedBigInt SUB(const UnsignedBigInt& a, const UnsignedBigInt& b)
    {
        unsigned int n = b.bits.size();
        unsigned int m = a.bits.size();
        UnsignedBigInt result;
        result.bits.clear();
        int c = 0;
        for (unsigned int i = 0; i < n; ++i)
        {
            int sum = c;
            sum += a.bits[i];
            sum -= b.bits[i];
            if (sum < 0)
            {
                result.bits.push_back(sum + 10);
                c = -1;
            }
            else
            {
                result.bits.push_back(sum);
```

```
47                  c = 0;
48              }
49          }
50          for(unsigned int i = n; i < m; ++i)
51          {
52              int sum = c;
53              sum += a.bits[i];
54              if (sum < 0)
55              {
56                  result.bits.push_back(sum + 10);
57                  c = -1;
58              }
59              else
60              {
61                  result.bits.push_back(sum);
62                  c = 0;
63              }
64          }
65          while(result.bits.back() == 0) result.bits.pop_back();
66          return result;
67      }
68      friend bool LESS(const UnsignedBigInt& num, const UnsignedBigInt& other)
69      {
70          if(num.bits.size() != other.bits.size())
71              return num.bits.size() < other.bits.size();
72
73          for(unsigned int i = num.bits.size() - 1; i >= 0; --i)
74          {
75              if (num.bits[i] != other.bits[i])
76                  return num.bits[i] < other.bits[i];
77          }
78          return false;
79      }
80  public:
81      UnsignedBigInt(void);
82      UnsignedBigInt(const std::string str);
83
84      UnsignedBigInt operator=(const std::string str);
85      UnsignedBigInt operator=(const char* str);
86      UnsignedBigInt operator=(int numer);
87
88      UnsignedBigInt operator+(const UnsignedBigInt& other) const;
89      UnsignedBigInt operator+=(const UnsignedBigInt& other);
90      UnsignedBigInt operator-(const UnsignedBigInt& other) const;
91
92      bool operator<(const UnsignedBigInt& other) const;
93      bool operator>(const UnsignedBigInt& other) const;
94      bool operator==(const UnsignedBigInt& other) const;
95      bool operator!=(const UnsignedBigInt& other) const;
96  private:
```

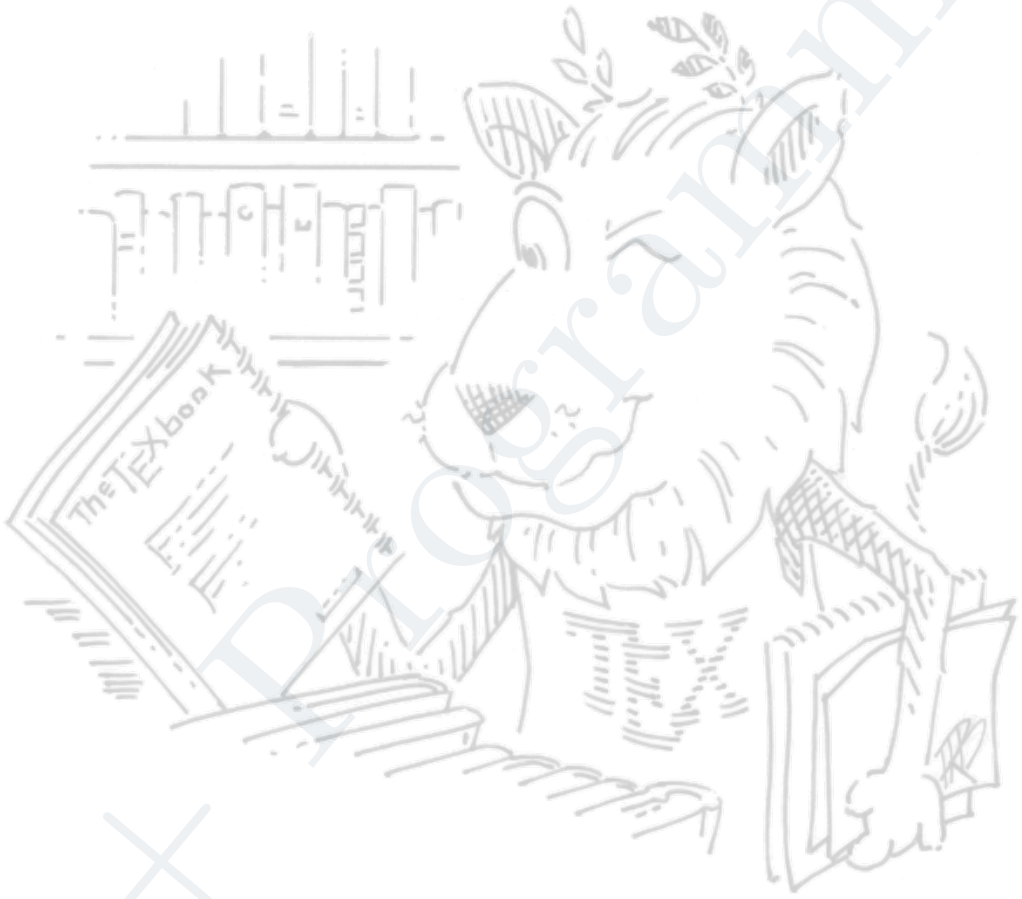
```
97      //vector参考: https://www.cplusplus.com/reference/vector/vector/?kw=vector
98      std::vector<int> bits; //bits[0]个位, bits[1]十位, ...
99      int sign;
100 };
101
102 UnsignedBigInt::UnsignedBigInt(void)
103 {
104     bits.push_back(0);
105     sign = 1;
106 }
107
108 UnsignedBigInt::UnsignedBigInt(const std::string str)
109 {
110     sign = 1;
111     if (str.empty())
112     {
113         bits.push_back(0);
114         return;
115     }
116
117     int index = 0;
118     //处理符号
119     if (str[index] == '-')
120     {
121         sign = -1;
122         index++;
123     }
124     else if (str[index] == '+')
125     {
126         index++;
127     }
128     else if (str[index] > '9' || str[index] < '0')
129     {
130         bits.push_back(0);
131         return;
132     }
133
134     for (int i = str.length() - 1; i >= index; --i)
135     {
136         if (str[i] > '9' || str[i] < '0') { bits.clear(); bits.push_back(0); return
                ; }
137         bits.push_back(str[i] - '0'); //vector没有push_front
138     }
139 }
140
141 UnsignedBigInt UnsignedBigInt::operator=(const std::string str)
142 {
143     *this = UnsignedBigInt(str);
144     return *this;
145 }
```

```
146
147 UnsignedBigInt UnsignedBigInt::operator=(const char* str)
148 {
149     *this = UnsignedBigInt(std::string(str));
150     return *this;
151 }
152
153 UnsignedBigInt UnsignedBigInt::operator=(int number)
154 {
155     std::string str = std::to_string(number);
156     *this = UnsignedBigInt(str);
157     return *this;
158 }
159
160 UnsignedBigInt UnsignedBigInt::operator+(const UnsignedBigInt& other)const
161 {
162     UnsignedBigInt result;
163     if(this->sign*other.sign == 1)
164     {//都是正数，或者负数
165         result = ADD(*this, other);
166         result.sign = this->sign;
167     }
168     else
169     {
170         if (LESS(*this, other))
171         {//第二个数大
172             result = SUB(other, *this);
173             if (this->sign == 1)
174                 result.sign = -1;
175             else
176                 result.sign = 1;
177         }
178         else
179         {//第一个数大
180             result = SUB(*this, other);
181             if (this->sign == 1)
182                 result.sign = 1;
183             else
184                 result.sign = -1;
185         }
186     }
187     return result;
188 }
189
190 UnsignedBigInt UnsignedBigInt::operator+=(const UnsignedBigInt& other)
191 {
192     UnsignedBigInt result = *this + other;
193     *this = result;
194     return result;
195 }
```

```
196
197 UnsignedBigInt UnsignedBigInt::operator-(const UnsignedBigInt& other)const
198 {
199     UnsignedBigInt result;
200     if (this->sign*other.sign == -1)
201     {
202         result = ADD(*this, other);
203         result.sign = this->sign;
204     }
205     else
206     {//都是正数，或者负数
207         if (LESS(*this, other))
208         {//第二个数大
209             result = SUB(other, *this);
210             if (this->sign == 1)
211                 result.sign = -1;
212             else
213                 result.sign = 1;
214         }
215         else
216         {//第一个数大
217             result = SUB(*this, other);
218             if (this->sign == 1)
219                 result.sign = 1;
220             else
221                 result.sign = -1;
222         }
223     }
224     return result;
225 }
226
227 bool UnsignedBigInt::operator<(const UnsignedBigInt & other)const
228 {
229     if (bits.size() != other.bits.size())
230         return bits.size() < other.bits.size();
231
232     for (unsigned int i = bits.size() - 1; i >= 0; --i)
233     {
234         if (bits[i] != other.bits[i])
235             return bits[i] < other.bits[i];
236     }
237     return false;
238 }
239
240 bool UnsignedBigInt::operator==(const UnsignedBigInt & other)const
241 {
242     if (bits.size() != other.bits.size())
243         return false;
244
245     for (unsigned int i = bits.size() - 1; i >= 0; --i)
```

```
246     {
247         if (bits[i] != other.bits[i])
248             return false;
249     }
250     return true;
251 }
252
253 bool UnsignedBigInt::operator!=(const UnsignedBigInt& other)const
254 {
255     return !(*this == other);
256 }
257
258 bool UnsignedBigInt::operator>(const UnsignedBigInt& other)const
259 {
260     return other < *this;
261 }
262
263 std::istream& operator >> (std::istream &is, UnsignedBigInt& bigint)
264 {
265     std::string bigint_str;
266     is >> bigint_str;
267     bigint = UnsignedBigInt(bigint_str);
268     return is;
269 }
270
271 std::ostream& operator<<(std::ostream &os, const UnsignedBigInt& bigint)
272 {
273     if (bigint.sign == -1)os << "-";
274     for (int i = bigint.bits.size() - 1; i >= 0; --i)
275         os << bigint.bits[i];
276     return os;
277 }
278
279 void Test()
280 {
281     UnsignedBigInt a, b, c;
282     a = UnsignedBigInt("-1000000000000000000");
283     b = UnsignedBigInt("99999999999999999999");
284     c = UnsignedBigInt("999");
285     std::cout << "a = " << a << ", b = " << b << std::endl;
286     std::cout << "a + b = " << a + b << std::endl;
287     std::cout << "a - b = " << a - b << std::endl;
288     std::cout << "b - a = " << b - a << std::endl;
289 }
290
291 int main()
292 {
293     UnsignedBigInt a, b, c;
294     a = "12349987654321999999";
295     c = 123456;
```

```
296     std::cout << c << std::endl;
297     while(std::cin >> b)
298     {
299         std::cout << a + b << std::endl;
300         std::cout << a - b << std::endl;
301         std::cout << (a < b) << std::endl;
302         a += b;
303         std::cout << a << std::endl;
304     }
305     return 0;
306 }
```



6 三维向量类

【描述】

```
构造一个三维向量类（class Vector），成员如下：
数据：分向量、向量模
函数：取模、显示向量
重载：加、自减、内积、外积、数乘、是否相等、是否共线、是否正交、拷贝、输入、输出

注意
重载cout输出流的输出格式：x y z（行末无空格）
成员函数void print()的输出格式：(x,y,z)

int main(int argc, char const *argv[])
{
    double a, b, c;
    std::cin >> a >> b >> c;
    Vector p1(a, b, c), p2, p3(p1), p4, x;
    p2 = p3;  //拷贝

    std::cin >> p4;

    p4.print(); //显示向量

    std::cout << p4.model() << std::endl;  //取模

    x = p1 + p4;
    std::cout << x << std::endl;

    p2 -= p4;
    std::cout << p2 << std::endl;

    x = p2*p3;
    std::cout << x << std::endl; //向量积

    std::cout << p2.a*p3.a + p2.b*p3.b + p2.c*p3.c << std::endl; //数量积

    int flag = p1 == p2;
    std::cout << flag << std::endl; //是否相等

    x = 1.5*p3;
    flag = p1 || x;
    std::cout << flag << std::endl; //是否平行（数乘）

    flag = (p1.a*p4.a + p1.b*p4.b + p1.c*p4.c) ? 0 : 1;
    std::cout << flag; //是否正交

    return 0;
}
```

【输入描述】

见样例

【输出描述】

见测试样例

【输入样例】

```
3 4 5
-1 7 -4
```

【输出样例】

```
(-1,7,-4)
8.12404
2 11 1
4 -3 9
-51 7 25
45
0
1
0
```

【提示】

main.cpp

```cpp
#include<iostream>
#include<cmath>

class Vector
{
public:
    Vector()
    {
        a = b = c = 0.0;
    }
    Vector(double a, double b, double c)
    {
        this->a = a;
        this->b = b;
        this->c = c;
    }
    Vector(const Vector& v)
    {
        this->a = v.a;
        this->b = v.b;
        this->c = v.c;
    }
    friend std::istream& operator>>(std::istream& in, Vector& cnt)
    {
        in >> cnt.a;
        in >> cnt.b;
        in >> cnt.c;
        return in;
    }
    friend std::ostream& operator<<(std::ostream& out, const Vector& cnt)
    {
        out << cnt.a << " " << cnt.b << " " << cnt.c;
        return out;
    }
    Vector operator+(const Vector& cnt)
    {
        Vector old;
        old.a = this->a + cnt.a;
        old.b = this->b + cnt.b;
        old.c = this->c + cnt.c;
        return old;
    }
    Vector& operator-=(const Vector& cnt)
    {
        this->a = this->a - cnt.a;
        this->b -= cnt.b;
        this->c -= cnt.c;
        return *this;
```

```
49      }
50      Vector operator*(const Vector& cnt)
51      {
52          Vector result;
53          result.a = this->b*cnt.c - cnt.b*this->c;
54          result.b = this->c*cnt.a - cnt.c*this->a;
55          result.c = this->a*cnt.b - cnt.a*this->b;
56          return result;
57      }
58      int operator==(const Vector& cnt)
59      {
60          return (this->a == cnt.a && this->b == cnt.b && this->c == cnt.c) ? 1 : 0;
61      }
62      friend Vector operator*(double num, const Vector& cnt)
63      {
64          Vector result;
65          result.a = num*cnt.a;
66          result.b = num*cnt.b;
67          result.c = num*cnt.c;
68          return result;
69      }
70      int operator||(const Vector& cnt)
71      {
72          double x = this->a / cnt.a;
73          if (abs(x - this->b / cnt.b)<10e-10 && abs(x - this->c / cnt.c)<10e-10)
74              return 1;
75          return 0;
76      }
77      double model()const
78      {
79          return sqrt(a*a + b*b + c*c);
80      }
81      void print()const
82      {
83          std::cout << "(" << this->a << "," << this->b << "," << this->c << ")" <<
                std::endl;
84      }
85      double a, b, c;
86  };
87
88  int main(int argc, char const *argv[])
89  {
90      double a, b, c;
91      std::cin >> a >> b >> c;
92      Vector p1(a, b, c), p2, p3(p1), p4, x;
93      p2 = p3;  //拷贝
94
95      std::cin >> p4;
96
97      p4.print(); //显示向量
```

```
98
99      std::cout << p4.model() << std::endl;  //取模
100
101     x = p1 + p4;
102     std::cout << x << std::endl;
103
104     p2 -= p4;
105     std::cout << p2 << std::endl;
106
107     x = p2*p3;
108     std::cout << x << std::endl; //向量积
109
110     std::cout << p2.a*p3.a + p2.b*p3.b + p2.c*p3.c << std::endl; //数量积
111
112     int flag = p1 == p2;
113     std::cout << flag << std::endl; //是否相等
114
115     x = 1.5*p3;
116     flag = p1 || x;
117     std::cout << flag << std::endl; //是否平行(数乘)
118
119     flag = (p1.a*p4.a + p1.b*p4.b + p1.c*p4.c) ? 0 : 1;
120     std::cout << flag; //是否正交
121
122     return 0;
123 }
```

7 Complex

【描述】

设计一个复数类 Complex，要求对运算符 “+” “-” “*” “/” 和 “+=” “-=” “*=” “/=” 进行重载，完成复数的加减乘除以及加减乘除复合赋值运算；并且重载 “«” 和 “»” 操作符完成复数的输入和输出。最后，重载 “==” 和 “!=” 比较两个复数是否相等。
使用以下 main 函数测试你的复数类

```
int main()
{
    Complex c1, c2;
    cin >> c1 >> c2;
    cout << "c1 = " << c1 << "\n" << "c2 = " << c2 << endl;
    cout << "c1+c2 = " << c1 + c2 << endl;
    cout << "c1-c2 = " << c1 - c2 << endl;
    cout << "c1*c2 = " << c1 * c2 << endl;
    cout << "c1/c2 = " << c1 / c2 << endl;
    cout << (c1 += c2) << endl;
    cout << (c1 -= c2) << endl;
    cout << (c1 *= c2) << endl;
    cout << (c1 /= c2) << endl;
    cout << (c1 == c2) << " " << (c1 != c2) << endl;

    return 0;
}
```

【输入描述】

输入有两行，每行输入两个表示复数 c1 和 c2 的浮点数。

【输出描述】

输出一共有 11 行，分别表示复数之间的各项操作，具体参见主函数和输出样例

【输入样例】

```
-4 6
2 5
```

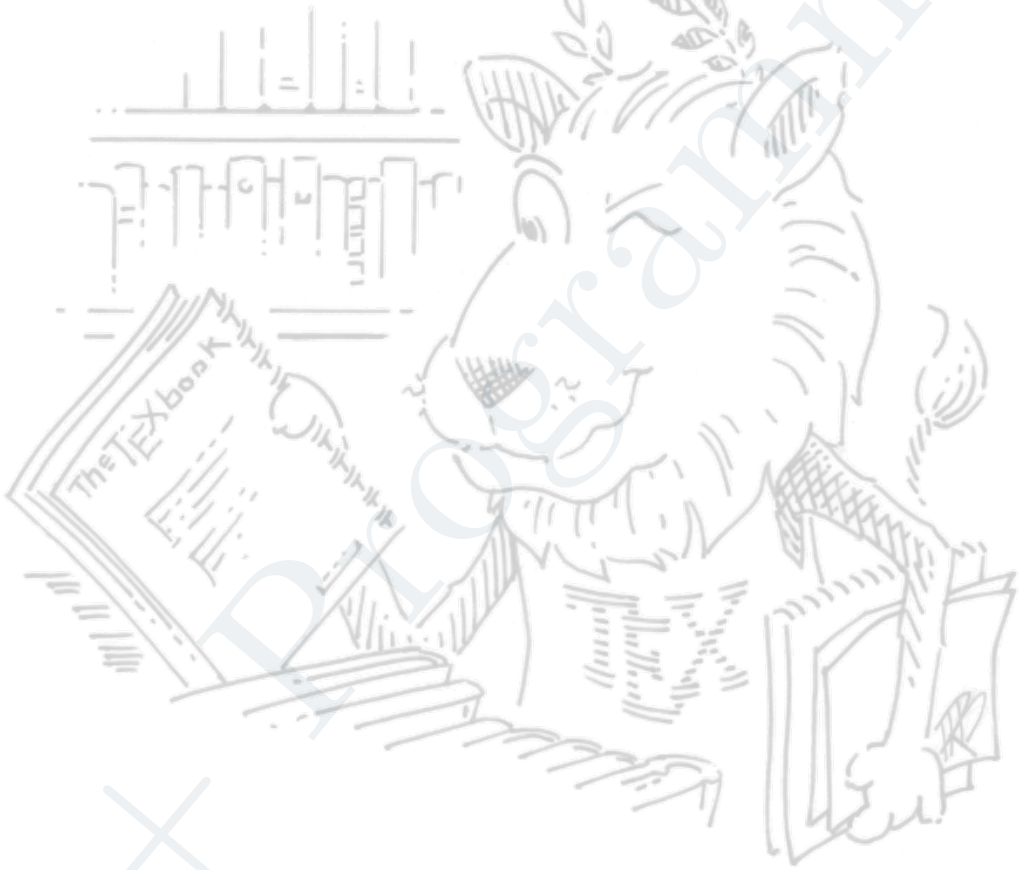
【输出样例】

```
c1 = -4 + 6i
c2 = 2 + 5i
c1+c2 = -2 + 11i
c1-c2 = -6 + 1i
c1*c2 = -38 + -8i
c1/c2 = 0.758621 + 1.10345i
```

```
-2 + 11i
-4 + 6i
-38 + -8i
-4 + 6i
0 1
```

【提示】

复数加法公式：(a+bi)+(c+di)=(a+c)+(b+d)i
复数减法公式：(a+bi)-(c+di)=(a-c)+(b-d)i
复数乘法公式：(a+bi)(c+di)=(ac-bd)+(ad+bc)i
复数除法公式：(a+bi)/(c+di)=[(ac+bd)/(cc+dd)]+[(bc-ad)/(cc+d*d)]i



main.cpp

```
#include<iostream>
#include<cstdlib>

class Complex
{
private:
    double x;
    double y;
public:
    Complex(double x = 0.0, double y = 0.0);
    Complex& operator+=(const Complex&);
    Complex& operator-=(const Complex&);
    Complex& operator*=(const Complex&);
    Complex& operator/=(const Complex&);
    friend Complex operator+(const Complex&, const Complex&);
    friend Complex operator-(const Complex&, const Complex&);
    friend Complex operator*(const Complex&, const Complex&);
    friend Complex operator/(const Complex&, const Complex&);
    friend bool operator==(const Complex&, const Complex&);
    friend bool operator!=(const Complex&, const Complex&);
    friend std::ostream& operator<<(std::ostream&, const Complex&);
    friend std::istream& operator>>(std::istream&, Complex&);
};

Complex::Complex(double x, double y) : x(x), y(y) {}

Complex& Complex::operator+=(const Complex& c)
{
    return *this = *this + c;
}

Complex& Complex::operator-=(const Complex& c)
{
    return *this = *this - c;
}

Complex& Complex::operator*=(const Complex& c)
{
    return *this = *this * c;
}

Complex& Complex::operator/=(const Complex& c)
{
    return *this = *this / c;
}

Complex operator+(const Complex& c1, const Complex& c2)
{
```

```
49      Complex temp;
50      temp.x = c1.x + c2.x;
51      temp.y = c1.y + c2.y;
52      return temp;
53  }
54
55  Complex operator-(const Complex& c1, const Complex& c2)
56  {
57      Complex temp;
58      temp.x = c1.x - c2.x;
59      temp.y = c1.y - c2.y;
60      return temp;
61  }
62
63  Complex operator*(const Complex& a, const Complex& b)
64  {
65      return Complex(a.x*b.x - a.y*b.y, a.x*b.y + b.x*a.y);
66  }
67
68  Complex operator/(const Complex& a, const Complex& c)
69  {
70      return Complex((a.x*c.x + a.y*c.y) / (c.x*c.x + c.y*c.y), (a.y*c.x - a.x*c.y) /
            (c.x*c.x + c.y*c.y));
71  }
72
73  bool operator==(const Complex& a, const Complex& b) { return ((a.x == b.x) && (a.y
        == b.y)); }
74
75  bool operator!=(const Complex& a, const Complex& b) { return !(a == b); }
76
77  std::ostream& operator<<(std::ostream& os, const Complex& c)
78  {
79      os << c.x << " + " << c.y << "i";
80      return os;
81  }
82
83  std::istream& operator>>(std::istream& is, Complex& c)
84  {
85      is >> c.x >> c.y;
86      return is;
87  }
88
89  int main()
90  {
91      Complex c1, c2;
92      std::cin >> c1 >> c2;
93      std::cout << "c1 = " << c1 << "\n" << "c2 = " << c2 << std::endl;
94      std::cout << "c1+c2 = " << c1 + c2 << std::endl;
95      std::cout << "c1-c2 = " << c1 - c2 << std::endl;
96      std::cout << "c1*c2 = " << c1 * c2 << std::endl;
```

```
97      std::cout << "c1/c2 = " << c1 / c2 << std::endl;
98      std::cout << (c1 += c2) << std::endl;
99      std::cout << (c1 -= c2) << std::endl;
100     std::cout << (c1 *= c2) << std::endl;
101     std::cout << (c1 /= c2) << std::endl;
102     std::cout << (c1 == c2) << " " << (c1 != c2) << std::endl;
103
104     return 0;
105 }
```

8 学生选课

【描述】

已知选课类 Subject 和学生类 Student 定义如下，学生类是选课类的友元类

```
class Subject    //选课类
{
private:
    double score[2]; //2门课成绩
    static const int SMath=4, SCpp=2; //2门课的学分，分别为4、2
public:
    Subject(int math = 0, int cpp = 0);
    void Input(); //输入2门课的成绩
    friend class Student; //友元类
};
class Student
{
private:
    std::string ID; //学号
    std::string name; //姓名
    double GPA; //平均学分积
    Subject sub; //选课，注意本题假设大家选课一样
public:
    Student(std::string id = "00000", std::string na = "Noname");
    void CalculateGPA(); //计算平均学分积
    void Input(); //输入学号和姓名
    void Show()const; //输出所有信息
};
```

请实现以上两个类，并用如下 main 函数进行测试:【注意，下述代码默认存在，提交代码时不能包括以下代码】

```
int main()
{
    int n; //学生人数
    std::cin >> n;
    Student *stu = new Student[n];
    for (int i = 0; i < n; i++)
    {
        stu[i].Input();
    }
    for (int i = 0; i < n; i++)
    {
        stu[i].CalculateGPA();
        stu[i].Show();
    }
    delete[] stu;

    return 0;
}
```

【输入描述】

第一行输入学生人数 n，然后依次输入 n 个学生的学号、姓名、2 门课的成绩

【输出描述】

见测试样例

【输入样例】

```
3
001
Mary
88
89
002
Bob
93
89
003
Lily
78
90
```

【输出样例】

```
ID: 001, Name: Mary
Math Cpp
88 89
GPA: 88.3333
ID: 002, Name: Bob
Math Cpp
93 89
GPA: 91.6667
ID: 003, Name: Lily
Math Cpp
78 90
GPA: 82
```

【提示】

输出时，冒号和逗号后各有一个空格，成绩后面有一个空格，课程名称之间有一个空格，“Cpp”后无空格

main.cpp

```cpp
#include<iostream>
#include<string>

class Subject    //选课类
{
private:
    double score[2]; //2门课成绩
    static const int SMath=4, SCpp=2; //2门课的学分，分别为4、2
public:
    Subject(int math = 0, int cpp = 0);
    void Input(); //输入2门课的成绩
    friend class Student; //友元类
};

Subject::Subject(int math, int cpp)
{
    score[0] = math;
    score[1] = cpp;
}

void Subject::Input()
{
    for (int i = 0; i < 2; i++)
        std::cin >> score[i];
}

class Student
{
private:
    std::string ID; //学号
    std::string name; //姓名
    double GPA; //平均学分积
    Subject sub; //选课，注意本题假设大家选课一样
public:
    Student(std::string id = "00000", std::string na = "Noname");
    void CalculateGPA(); //计算平均学分积
    void Input(); //输入学号和姓名
    void Show()const; //输出所有信息
};

Student::Student(std::string id, std::string na)
{
    ID = id;
    name = na;
    GPA = 0;
}

void Student::CalculateGPA() //计算平均学分积
```

```
49  {
50      GPA = (sub.score[0] * sub.SMath + sub.score[1] * sub.SCpp)
51          / (sub.SMath + sub.SCpp);
52  }
53
54  void Student::Input()
55  {
56      std::cin >> ID >> name;
57      sub.Input();
58  }
59
60  void Student::Show()const //输出所有信息
61  {
62      std::cout << "ID: " << ID << ", Name: " << name << std::endl;
63      std::cout << "Math Cpp"<< std::endl;
64      std::cout << sub.score[0] << " " << sub.score[1] << std::endl;
65      std::cout << "GPA: " << GPA << std::endl;
66  }
67
68  int main()
69  {
70      int n; //学生人数
71      std::cin >> n;
72      Student *stu = new Student[n];
73
74      for (int i = 0; i < n; i++)
75      {
76          stu[i].Input();
77      }
78
79      for (int i = 0; i < n; i++)
80      {
81          stu[i].CalculateGPA();
82          stu[i].Show();
83      }
84
85      delete[] stu;
86      return 0;
87  }
```