

时间类:

```
1.     #include <iostream>
2.     using namespace std;
3.     class Cmytime{
4.     private:
5.         int _h, _m, _s;
6.     public:
7.         Cmytime(int h = 0, int m = 0, int s = 0){
8.             _h = h, _m = m; _s = s;
9.         }
10.        int Set(int h,int m,int s){
11.            if(h<0 || h>23 || m<0 || m>59 || s<0 || s>59){
12.
13.                return 0;
14.            }
15.            _h = h, _m = m; _s = s;
16.            return 1;
17.        }
18.        void Show(){
19.            cout<<_h<<":"<<_m<<":"<<_s;
20.        }
21.        void AddOneSecond(){
22.            _s += 1;
23.            if(_s == 60){
24.                _s = 0;
25.                _m += 1;
26.            }
27.            if(_m == 60){
28.                _m = 0;
29.                _h += 1;
30.            }
31.            if(_h == 24){
32.                _h = 0;
33.            }
34.        }
35.        int AddNSeconds(int n){
36.            _s += n;
37.            if(_s >= 60){
38.                int muint = _s / 60;
39.                _s %= 60;
40.                _m += muint;
41.            }
42.            if(_m >= 60){
43.                int h_P = _m / 60;
```

```

44.     _m %= 60;
45.     _h += h_P;
46. }
47. if(_h >= 24){
48.     int day_p = _h / 24;
49.     _h %= 24;
50.     return day_p;
51. }
52. return 0;
53. }
54. };
55. //StudybarCommentBegin
56. int main(void) {
57.     int h,m,s;
58.     cin>>h>>m>>s;
59.     Cmytime t1(3,2,1),t2,t3(5);
60.     t1.Show();
61.     cout<<"\n";
62.     t1.Set(h,m,s);
63.     t1.Show();
64.     cout<<"\n";
65.     t2.Show();
66.     cout<<"\n";
67.     t3.Show();
68.     return 0;
69. }
70. //StudybarCommentEnd

```

时间类综合：

```

1.     //StudybarCommentBegin
2.     #include <iostream>
3.     #include <iomanip>
4.     using std::cin;
5.     using std::cout;
6.     using std::endl;
7.     using std::setfill;
8.     using std::setw;
9.     //StudybarCommentEnd
10.
11.     class Time{
12.     private:
13.         int _hour;
14.         int _minute;
15.         int _second;

```

```
16.
17.     friend Time operator+(const int n, const Time t);
18.     friend Time operator-(const Time t, const int n);
19. public:
20.     Time(int hour = 0, int minute=0, int second=0){
21.         _hour = hour, _minute = minute, _second = second;
22.         setStandard();
23.         setStandardMinus();
24.     }
25.     void setTime(int hour, int minute, int second){
26.         _hour = hour, _minute = minute, _second = second;
27.         setStandard();
28.         setStandardMinus();
29.     }
30.     void setStandard(){
31.         int mod = 0;
32.         if(_second >= 60){
33.             mod = _second / 60;
34.             _second %= 60;
35.         }
36.         _minute += mod;
37.         if(_minute >= 60){
38.             mod = _minute / 60;
39.             _minute %= 60;
40.         }
41.         else{
42.             mod = 0;
43.         }
44.         _hour += mod;
45.         if(_hour >= 24){
46.             mod = _hour / 24;
47.             _hour %= 24;
48.         }
49.     }
50.     void setStandardMinus(){
51.         int mod = 0;
52.         if(_second < 0){
53.             mod = (-_second + 59) / 60; //向上取整
54.             _second = _second%60 + 60; //向上取整
55.         }
56.         _minute -= mod;
57.         if(_minute < 0 ){
58.             mod = (-_minute + 59) / 60;
59.             _minute = _minute%60 + 60;
```

```

60.     }
61.     else{
62.         mod = 0;
63.     }
64.     _hour -= mod;
65.     if(_hour < 0){
66.         _hour = _hour % 24 + 24;
67.     }
68.     }
69.     Time& operator++(){
70.         ++_second;
71.         setStandard();
72.         return *this;
73.     }
74.     Time operator++(int){
75.         Time tmp = *this;
76.         ++_second;
77.         setStandard();
78.         return tmp;
79.     }
80.
81.     Time& operator--(){
82.         --_second;
83.         setStandardMinus();
84.         return *this;
85.     }
86.     Time operator--(int){
87.         Time tmp = *this;
88.         --_second;
89.         setStandardMinus();
90.         return tmp;
91.     }
92.     void printTime();
93. };
94.
95. void Time::printTime()
96. {
97.     cout<<setfill('0')<<setw(2)<<_hour
98.     <<":"<<setw(2)<<_minute<<":"
99.     <<setw(2)<<_second<<endl;
100. }
101. Time operator+(const int n, const Time t){
102.     Time tmp = t;
103.     tmp._second += n;

```

```
104.     tmp.setStandard();
105.     tmp.setStandardMinus();
106.     return tmp;
107. }
108. Time operator-(const Time t, const int n){
109.     Time tmp = t;
110.     tmp._second -= n;
111.     tmp.setStandardMinus();
112.     tmp.setStandard();
113.     return tmp;
114. }
```

GradeBook 类:

```
1.    #include <iostream>
2.    using namespace std;
3.    class GradeBook{
4.    private:
5.        string _course_name;
6.        string _instructor_name;
7.    public:
8.        GradeBook(const string& courseName, const string& structorName){
9.            _course_name = courseName, _instructor_name = structorName;
10.       }
11.       string getInstructorName(){
12.           return _instructor_name;
13.       }
14.       void setInstructorName(const string& structorName){
15.           _instructor_name = structorName;
16.       }
17.       void displayMessage(){
18.           cout<<"Welcome to the grade book for"<<endl<<_course_name<<"!"<<endl;
19.           cout<<"This course is presented by: "<<_instructor_name<<endl;
20.       }
21.    };
```

Account 类:

```
1.    #include <iostream>
2.    using namespace std;
3.
4.    class Account{
5.    public:
6.        double _Balance;
7.        explicit Account(double balance){
8.            if(balance < 0){
9.                cout<<"\nError:Initial balance cannot be negative.\n"<<endl;
10.               balance = 0;
11.            }
12.            _Balance = balance;
13.        }
14.        void credit(double money){
15.            _Balance += money;
16.        }
17.        void debit(double money){
18.            if(money <= _Balance){
19.                _Balance -= money;
20.            }
```

```

21.     else
22.     {cout<<"Debit amount exceeded account balance.\n"<<endl; }
23.     }
24.     double getBalance(){
25.         return _Balance;
26.     }
27. };

```

Invoice 类:

```

1.     #include <iostream>
2.     using namespace std;
3.     class Invoice{
4.     private:
5.         string _part_number;
6.         string _part_description;
7.         int _quantity;
8.         int _price_per_item;
9.     public:
10.        Invoice(string part_number = "111", string part_description = "None", int q
            uantity = 0, int price_per_item = 0){
11.            if(quantity<0){
12.                quantity = 0;
13.            }
14.            if(price_per_item<0){
15.                price_per_item = 0;
16.            }
17.            _part_number = part_number, _part_description = part_description;
18.            _quantity = quantity;
19.            _price_per_item = price_per_item;
20.        }
21.        string getPartNumber(){
22.            return _part_number;
23.        }
24.        string getPartDescription(){
25.            return _part_description;
26.        }
27.        int getQuantity(){
28.            return _quantity;
29.        }
30.        int getPricePerItem(){
31.            return _price_per_item;
32.        }
33.        int getInvoiceAmount(){
34.            return _quantity*_price_per_item;

```

```

35.     }
36.     void setPartNumber(const string& part_number){
37.         _part_number = part_number;
38.     }
39.     void setPartDescription(const string& part_description){
40.         _part_description = part_description;
41.     }
42.     void setQuantity(const int& quantity){
43.         if(quantity<0){
44.             _quantity = 0;
45.             cout<<"quantity set to 0";
46.         }
47.         else{
48.             _quantity = quantity;
49.         }
50.     }
51.     void setPricePerItem(const int& price_per_item){
52.         if(price_per_item < 0){
53.             _price_per_item = 0;
54.         }
55.         else{
56.             _price_per_item = price_per_item;
57.         }
58.     }
59. };
60. //StudybarCommentBegin
61. int main()
62. {
63.     // create an Invoice object
64.     Invoice invoice( "12345", "Hammer", 100, 5 );
65.     // display the invoice data members and calculate the amount
66.     cout << "Part number: " << invoice.getPartNumber() << endl;
67.     cout << "Part description: " << invoice.getPartDescription() << endl;
68.     cout << "Quantity: " << invoice.getQuantity() << endl;
69.     cout << "Price per item: $" << invoice.getPricePerItem() << endl;
70.     cout << "Invoice amount: $" << invoice.getInvoiceAmount() << endl;
71.     // modify the invoice data members
72.     invoice.setPartNumber( "123456" );
73.     invoice.setPartDescription( "Saw" );
74.     invoice.setQuantity( -5 ); // negative quantity, so quantity set to 0
75.     invoice.setPricePerItem( 10 );
76.     cout << "\nInvoice data members modified.\n";
77.     // display the modified invoice data members and calculate new amount
78.     cout << "Part number: " << invoice.getPartNumber() << endl;

```



```
79.         cout << "Part description: " << invoice.getPartDescription() << endl;
80.         cout << "Quantity: " << invoice.getQuantity() << endl;
81.         cout << "Price per item: $" << invoice.getPricePerItem() << endl;
82.         cout << "Invoice amount: $" << invoice.getInvoiceAmount() << endl;
83.         return 0; // indicate successful termination
84.     } // end main
85.     //StudybarCommentEnd
```

分数类:

```
1.  #include <iostream>
2.  #include <cmath>
3.  #include <string>
4.  using namespace std;
5.
6.  int gcd(int a, int b){
7.      a = a < 0 ? -a : a;
8.      b = b < 0 ? -b : b;
9.      if( b > a){
10.         int tmp = a;
11.         a = b;
12.         b = tmp;
13.     }
14.     while (b != 0) {
15.         int remainder = a % b;
16.         a = b;
17.         b = remainder;
18.     }
19.     return a;
20. }
21. int lcm(int a, int b) {
22.     if (a == 0 || b == 0) return 0;
23.     return (a * b) / gcd(a, b);
24. }
25. class Fraction{
26. private:
27.     int _mole; //分子
28.     int _deno; //分母
29.
30.     friend void fswap(Fraction& a, Fraction& b);
31.     friend void printtest(Fraction& a);
32.     friend ostream& operator<<(ostream& os, const Fraction& x);
33.     friend istream& operator>>(istream& in, Fraction& x);
34. public:
35.     Fraction(int mole = 1, int deno = 1){
36.         //处理分子分母负号问题
37.         if(mole*deno<0){
38.             mole = -abs(mole);
39.             deno = abs(deno);
40.         }
41.         else if(mole<0 && deno<0){
42.             mole = abs(mole);
43.             deno = abs(deno);
```

```

44.     }
45.     // 处理最简分数
46.     int gcbNum = gcd(mole, deno);
47.     mole /= gcbNum;
48.     deno /= gcbNum;
49.     // 赋值
50.     _mole = mole;
51.     _deno = deno;
52. }
53. Fraction(double x){
54.     int sign = 1;
55.     while(int(x)!=x){
56.         x *= 10;
57.         sign *= 10;
58.     }
59.     int gcbNum = gcd(int(x), sign);
60.     x /= gcbNum;
61.     sign /= gcbNum;
62.     // 赋值
63.     _mole = int(x);
64.     _deno = sign;
65. }
66. int getMole(){
67.     return _mole;
68. }
69. int getDeno(){
70.     return _deno;
71. }
72. void Show(){
73.     cout<<_mole<<"/"<<_deno;
74. }
75. double to_double(){
76.     return _mole*1.0/_deno;
77. }
78.
79. Fraction operator-(const Fraction& b){
80.     int deno = lcm(_deno, b._deno);
81.     int mole = _mole*(deno/_deno) - b._mole*(deno/b._deno);
82.     return Fraction(mole, deno);
83. }
84. Fraction operator+(const Fraction& b){
85.     int deno = lcm(_deno, b._deno);
86.     int mole = _mole*(deno/_deno) + b._mole*(deno/b._deno);
87.     return Fraction(mole, deno);

```

```

88.     }
89.     Fraction operator*(const Fraction& b){
90.         return Fraction(_mole*b._mole, _deno*b._deno);
91.     }
92.     Fraction operator/(const Fraction& b){
93.         return Fraction(_mole*b._deno, _deno*b._mole);
94.     }
95.     bool operator==(const Fraction& b ){
96.         return _mole==b._mole && _deno==b._deno;
97.     }
98.     bool operator!=(const Fraction& b ){
99.         return !(*this==b);
100.    }
101.    bool operator>(const Fraction& b){
102.        Fraction c = b;
103.        return to_double() > c.to_double();
104.    }
105.    bool operator<=(const Fraction& b){
106.        return !(*this > b);
107.    }
108.    bool operator<(const Fraction& b){
109.        Fraction c = b;
110.        return to_double() < c.to_double();
111.    }
112.    bool operator>=(const Fraction& b){
113.        return !(*this < b);
114.    }
115. };
116. void fswap(Fraction& a, Fraction& b){
117.     Fraction tmp = a;
118.     a = b;
119.     b = tmp;
120. }
121. ostream& operator<<(ostream& os, const Fraction& x){
122.     os<<x._mole<<"/"<<x._deno;
123.     return os;
124. }
125. istream& operator>>(istream& in, Fraction& x) {
126.     int mole, deno;
127.     in >> mole >> deno;
128.     Fraction temp(mole, deno); // 通过构造函数处理符号和约分
129.     x._mole = temp.getMole();
130.     x._deno = temp.getDeno();
131.     return in;}

```

复数集合类:

```
1.     #include<iostream>
2.     #include<vector>
3.     #include<algorithm>
4.     #include<cmath>
5.     using namespace std;
6.
7.     class Cmycomplex {
8.     private:
9.         double _real;
10.        double _imag;
11.        friend Cmycomplex operator+(const Cmycomplex& cplx, double x);
12.        friend Cmycomplex operator+(double x, const Cmycomplex& cplx);
13.        friend std::ostream& operator<<(std::ostream& os, const Cmycomplex& cplx);
14.        friend std::istream& operator>>(std::istream& is, Cmycomplex& cplx);
15.
16.    public:
17.        Cmycomplex(double real = 0, double imag = 0) : _real(real), _imag(imag) {}
18.
19.        double GetReal() const { return _real; }
20.        double GetImaginary() const { return _imag; }
21.
22.        // 模计算
23.        double modulus() const {
24.            return sqrt(_real * _real + _imag * _imag);
25.        }
26.
27.        bool operator==(const Cmycomplex& other) const {
28.            return _real == other._real && _imag == other._imag;
29.        }
30.
31.        void Set(double real, double imag) {
32.            _real = real;
33.            _imag = imag;
34.        }
35.    };
36.    Cmycomplex operator+(const Cmycomplex& cplx, double x){
37.        return Cmycomplex(cplx._real+x, cplx._imag);
38.    }
39.    Cmycomplex operator+(double x, const Cmycomplex& cplx){
40.        return Cmycomplex(cplx._real+x, cplx._imag);
41.    }
42.    std::istream& operator>>(std::istream& is, Cmycomplex& cplx) {
43.        is >> cplx._real >> cplx._imag;
```

```

44.         return is;
45.     }
46.     std::ostream& operator<<(std::ostream& os, const Cmycomplex& cplx) {
47.         os << "(" << cplx._real;
48.         if (cplx._imag != 0) {
49.             os << (cplx._imag > 0 ? "+" : "") << cplx._imag << "i";
50.         }
51.         os << ")";
52.         return os;
53.     }
54.
55.     class Cassemblage {
56.     private:
57.         vector<Cmycomplex> elements;
58.     public:
59.         Cassemblage() = default;
60.         void Set(Cmycomplex arr[], int n) {
61.             elements.clear();
62.             for (int i = 0; i < n; ++i) {
63.                 if (find(elements.begin(), elements.end(), arr[i]) == elements.end()) {
64.                     elements.push_back(arr[i]);
65.                 }
66.             }
67.         }
68.         Cassemblage operator+(const Cassemblage& other) const {
69.             Cassemblage result = *this;
70.             for (const auto& e : other.elements) {
71.                 if (find(result.elements.begin(), result.elements.end(), e) == result.elements.end()) {
72.                     result.elements.push_back(e);
73.                 }
74.             }
75.             return result;
76.         }
77.         Cassemblage operator&(const Cassemblage& other) const {
78.             Cassemblage result;
79.             for (const auto& e : elements) {
80.                 if (find(other.elements.begin(), other.elements.end(), e) != other.elements.end()) {
81.                     result.elements.push_back(e);
82.                 }
83.             }
84.             return result;
85.         }
86.         Cassemblage operator-(const Cassemblage& other) const {
87.             Cassemblage result;

```

```
88.         for (const auto& e : elements) {
89.             if (find(other.elements.begin(), other.elements.end(), e) == other.elements.end()) {
90.                 result.elements.push_back(e);
91.             }
92.         }
93.         return result;
94.     }
95.     void Show() const {
96.         if (elements.empty()) {
97.             cout << "empty";
98.             return;
99.         }
100.        vector<Cmycomplex> sorted = elements;
101.        sort(sorted.begin(), sorted.end(), [](const Cmycomplex& a, const Cmycomplex& b) {
102.            return a.modulus() < b.modulus();
103.        });
104.        for (size_t i = 0; i < sorted.size(); ++i) {
105.            cout << sorted[i] << (i != sorted.size()-1 ? " " : "");
106.        }
107.    }
108.};
```

复数类:

```
1.  #include<iostream>
2.  #include<iomanip>
3.  #include <sstream>
4.  #include <cmath>
5.  using namespace std;
6.
7.  class Cmycomplex{
8.  private:
9.      double _real;
10.     double _imag;
11.     friend Cmycomplex operator+(const Cmycomplex& cplx, double x);
12.     friend Cmycomplex operator+(double x, const Cmycomplex& cplx);
13.     friend std::ostream& operator<<(std::ostream& os, const Cmycomplex& cplx);
14.     friend std::istream& operator>>(std::istream& is, Cmycomplex& cplx);
15. public:
16.     Cmycomplex(double real = 0, double imag = 0){
17.         _real = real;
18.         _imag = imag;
19.     }
20.     void Show(){
21.         cout <<setiosflags(ios::fixed);
22.         if(_imag<0){
23.             cout<<"("<<setprecision(2)<<_real<<setprecision(2)<<_imag<<"i)"<<endl;
24.         }
25.         else if(_imag==0){
26.             cout<<setprecision(2)<<_real<<endl;
27.         }
28.         else{
29.             cout<<"("<<setprecision(2)<<_real<<"+"<<setprecision(2)<<_imag<<"i)"<<endl;
30.         }
31.     }
32.     // void Show(){
33.     //     if(_imag<0){
34.     //         cout<<"("<<_real<<_imag<<"i)"<<endl;
35.     //     }
36.     //     else{
37.     //         cout<<"("<<_real<<"+"<<_imag<<"i)"<<endl;
38.     //     }
39.     // }
40.     double GetReal(){
41.         return _real;
42.     }
```



```

43.     double GetImaginary(){
44.         return _imag;
45.     }
46.     Cmycomplex Add(const Cmycomplex& x){
47.         _real += x._real;
48.         _imag += x._imag;
49.         return *this;
50.     }
51.     Cmycomplex operator+(const Cmycomplex& x){
52.         return Cmycomplex(_real+x._real, _imag+x._imag);
53.     }
54.     Cmycomplex operator-(const Cmycomplex& x){
55.         return Cmycomplex(_real-x._real, _imag-x._imag);
56.     }
57.     Cmycomplex operator*(const Cmycomplex& x){
58.         return Cmycomplex(_real*x._real-_imag*x._imag, _real*x._imag+_imag*x._real)
59.         ;
60.     }
61.     Cmycomplex operator/(const Cmycomplex& x){
62.         double newReal = (_real*x._real+_imag*x._imag)/(x._real*x._real + x._imag*
63.         x._imag);
64.         double newImag = (_imag*x._real-_real*x._imag)/(x._real*x._real + x._imag*
65.         x._imag);
66.         return Cmycomplex(newReal, newImag);
67.     }
68.     void Set(double real, double imag){
69.         _real = real;
70.         _imag = imag;
71.     }
72.     bool operator==(const Cmycomplex& x){
73.         return _real==x._real && _imag == x._imag;
74.     }
75.     bool operator!=(const Cmycomplex& x){
76.         return _real!=x._real || _imag != x._imag;
77.     }
78.     Cmycomplex sqrt(){
79.         double newReal = std::sqrt((_real+std::sqrt(_real*_real+_imag*_imag))/2);
80.         double newImag = _imag/std::sqrt(2*_real+2*std::sqrt(_real*_real+_imag*_im
81.         ag));
82.         return Cmycomplex(newReal, newImag);
83.     }
84. };
85. Cmycomplex operator+(const Cmycomplex& cplx, double x){
86.     return Cmycomplex(cplx._real+x, cplx._imag);

```

```

83.     }
84.     Cmycomplex operator+(double x, const Cmycomplex& cplx){
85.         return Cmycomplex(cplx._real+x, cplx._imag);
86.     }
87.     std::ostream& operator<<(std::ostream& os, const Cmycomplex& cplx) {
88.         os<<std::fixed<<setprecision(2);
89.         if(cplx._imag<0){
90.             os<<cplx._real<<cplx._imag<<"i";
91.         }
92.         else if(cplx._imag==0){
93.             os<<cplx._real;
94.         }
95.         else{
96.             os<<cplx._real<<"+"<<cplx._imag<<"i";
97.         }
98.         return os;
99.     }
100.    std::istream& operator>>(std::istream& is, Cmycomplex& cplx) {
101.        string s;
102.        cin>>s;
103.        stringstream ss(s);
104.        if(s.find('i')!=string::npos){
105.            //有虚部
106.            if(s.find('+', 1)!=string::npos||s.find('-', 1)!=string::npos){
107.                // 复数
108.                ss>>cplx._real>>cplx._imag;
109.                ss.get(); // 消耗 i
110.            }
111.            else{
112.                // 纯虚数
113.                cplx._real = 0.0;
114.                ss>>cplx._imag;
115.                ss.get();// 消耗 i
116.            }
117.        }
118.        else{
119.            // 退化为实数
120.            cplx._imag = 0.0;
121.            ss>>cplx._real;
122.        }
123.        return is;
124.    }
125.    typedef Cmycomplex ComplexNumber;

```

复数求二次函数：

```
1.     #include <iostream>
2.     #include <cmath>
3.     using namespace std;
4.
5.     typedef struct {
6.         double real;
7.         double imag;
8.     } Complex;
9.     // 复数加法
10.    Complex add(Complex a, Complex b) {
11.        return (Complex){a.real + b.real, a.imag + b.imag};
12.    }
13.    // 复数减法
14.    Complex sub(Complex a, Complex b) {
15.        return (Complex){a.real - b.real, a.imag - b.imag};
16.    }
17.    // 复数乘法
18.    Complex mul(Complex a, Complex b) {
19.        double real = a.real * b.real - a.imag * b.imag;
20.        double imag = a.real * b.imag + a.imag * b.real;
21.        return (Complex){real, imag};
22.    }
23.    // 复数除法
24.    Complex div(Complex a, Complex b) {
25.        double denominator = b.real * b.real + b.imag * b.imag;
26.        double real = (a.real * b.real + a.imag * b.imag) / denominator;
27.        double imag = (a.imag * b.real - a.real * b.imag) / denominator;
28.        return (Complex){real, imag};
29.    }
30.    // 复数平方根
31.    Complex sqrt_complex(Complex z) {
32.        double r = sqrt(z.real * z.real + z.imag * z.imag);
33.        double theta = atan2(z.imag, z.real);
34.        double sqrt_r = sqrt(r);
35.        double angle = theta / 2;
36.        return (Complex){sqrt_r * cos(angle), sqrt_r * sin(angle)};
37.    }
38.    // 格式化输出复数
39.    void print_complex(Complex c) {
40.        if (c.imag >= 0)
41.            printf("%.21f+%.21fi\n", c.real, c.imag);
42.        else
43.            printf("%.21f-%.21fi\n", c.real, -c.imag);
```

```
44.     }
45.     int main() {
46.         Complex a, b, c;
47.         cin>>a.real>>a.imag;
48.         cin>>b.real>>b.imag;
49.         cin>>c.real>>c.imag;
50.         Complex D = sub(mul(b, b), mul(mul((Complex){4, 0}, a), c));
51.         Complex sqrtD = sqrt_complex(D);
52.         Complex root1 = div(add(sub((Complex){0, 0}, b), sqrtD), mul((Complex){2, 0}, a));
53.         Complex root2 = div(sub(sub((Complex){0, 0}, b), sqrtD), mul((Complex){2, 0}, a));
54.         if (root2.imag > root1.imag) {
55.             Complex temp = root1;
56.             root1 = root2;
57.             root2 = temp;
58.         }
59.         print_complex(root1);
60.         print_complex(root2);
61.         return 0;
62.     }
```

长方形类:

```
1.     #include <iostream>
2.     #include <algorithm>
3.     #include <cmath>
4.     #include <cstdlib>
5.     using namespace std;
6.
7.     class Rectangle {
8.     private:
9.         double points_[4][2];
10.        double length_;
11.        double width_;
12.
13.        void SetPoints(const double rect[4][2]) {
14.            // 调整坐标并验证有效性
15.            for (int i = 0; i < 4; ++i) {
16.                // 处理横坐标
17.                if (rect[i][0] < 0 || rect[i][0] > 20.0) {
18.                    cout << "第" << (i+1) << "个点的横坐标值无效,被置为 0" << endl;
19.                    points_[i][0] = 0;
20.                } else {
21.                    points_[i][0] = rect[i][0];
22.                }
23.                // 处理纵坐标
24.                if (rect[i][1] < 0 || rect[i][1] > 20.0) {
25.                    cout << "第" << (i+1) << "个点的纵坐标值无效,被置为 0" << endl;
26.                    points_[i][1] = 0;
27.                } else {
28.                    points_[i][1] = rect[i][1];
29.                }
30.            }
31.            // 定义点结构并排序
32.            struct Point {
33.                double x, y;
34.                bool operator<(const Point& other) const {
35.                    return (y != other.y) ? (y > other.y) : (x < other.x);
36.                }
37.            };
38.            Point sorted[4];
39.            for (int i = 0; i < 4; ++i) {
40.                sorted[i] = {points_[i][0], points_[i][1]};
41.            }
42.            sort(sorted, sorted + 4);
43.            // 验证四边形基本属性
```

```
44.     const bool top_pair = (sorted[0].y == sorted[1].y);
45.     const bool bottom_pair = (sorted[2].y == sorted[3].y);
46.     const bool vertical_edges = (sorted[0].x == sorted[2].x) &&
47.                                   (sorted[1].x == sorted[3].x);
48.     const bool horizontal_edges = ((sorted[1].x - sorted[0].x) ==
49.                                     (sorted[3].x - sorted[2].x));
50.     const bool vertical_length = ((sorted[0].y - sorted[2].y) ==
51.                                    (sorted[1].y - sorted[3].y));
52.     if (!(top_pair && bottom_pair && vertical_edges &&
53.           horizontal_edges && vertical_length)) {
54.         cout << "不能构成长方形! " << endl;
55.         exit(0);
56.     }
57.     // 计算长宽
58.     const double edge_x = sorted[1].x - sorted[0].x;
59.     const double edge_y = sorted[0].y - sorted[2].y;
60.     length_ = fmax(edge_x, edge_y);
61.     width_ = fmin(edge_x, edge_y);
62. }
63. public:
64.     explicit Rectangle(const double rect[4][2]) {
65.         SetPoints(rect);
66.     }
67.     double length() const { return length_; }
68.     double width() const { return width_; }
69.     double perimeter() const { return 2 * (length_ + width_); }
70.     double area() const { return length_ * width_; }
71.     bool square() const { return length_ == width_; }
72. };
```

24/12 小时制时间:

```
1.     #include <iostream>
2.     #include <iomanip>
3.     using namespace std;
4.
5.     class Clock24 {
6.     private:
7.         int _hour,_minute,_second;
8.     public:
9.         bool IS24;
10.        Clock24(int h=0,int m=0,int s=0,bool is24=true):IS24(is24),_hour(h%24),_minute(m%60),_second(s%60) {}
11.
12.        void setStandard(){
13.            int mod = 0;
14.            if(_second >= 60){
15.                mod = _second / 60;
16.                _second %= 60;
17.            }
18.            _minute += mod;
19.            if(_minute >= 60){
20.                mod = _minute / 60;
21.                _minute %= 60;
22.            }
23.            else{
24.                mod = 0;
25.            }
26.            _hour += mod;
27.            if(_hour >= 24){
28.                mod = _hour / 24;
29.                _hour %= 24;
30.            }
31.        }
32.        void setStandardMinus(){
33.            int mod = 0;
34.            if(_second < 0){
35.                mod = (-_second + 59)/ 60; // 向上取整
36.                _second = _second%60 + 60; // 向上取整
37.            }
38.            _minute -= mod;
39.            if(_minute < 0 ){
40.                mod = (-_minute + 59) / 60;
41.                _minute = _minute%60 + 60;
42.            }
```

```

43.     else{
44.         mod = 0;
45.     }
46.     _hour -= mod;
47.     if(_hour < 0){
48.         _hour = _hour % 24 + 24;
49.     }
50. }
51. Clock24& operator++(){
52.     ++_second;
53.     setStandard();
54.     return *this;
55. }
56. Clock24 operator++(int){
57.     Clock24 tmp = *this;
58.     ++_second;
59.     setStandard();
60.     return tmp;
61. }
62. friend istream& operator>>(istream & in,Clock24 &c);
63. friend ostream& operator<<(ostream & out,const Clock24 &c);
64. int getSecondsOfDay()const{
65.     return _second + _minute*60+_hour*3600;
66. }
67. };
68. ostream& operator<<(ostream & out,const Clock24 &c)
69. {
70.     if (c.IS24)
71.         out<<setfill('0')<<setw(2)<<c._hour<<":"<<setw(2)<<c._minute<<":"<<setw(2)<<c._second;
72.     else
73.         out<<setfill('0')<<setw(2)<< (c._hour%12==0? 12:c._hour%12 )<<":"<<setw(2)<<c._minute<<":"
<<setw(2)<<c._second << (c._hour>=12?" PM":" AM");
74.     return out;
75. }
76. istream& operator>>(istream & in,Clock24 &c){
77.     in>>c._hour>>c._minute>>c._second;
78.     c.setStandard();
79.     c.setStandardMinus();
80.     return in;
81. }

```


日期类:

```
1. //StudybarCommentBegin
2. #include <iostream>
3. #include <iomanip>
4. using std::cin;
5. using std::cout;
6. using std::endl;
7. using std::setfill;
8. using std::setw;
9. //StudybarCommentEnd
10.
11. class Time{
12. private:
13.     int _hour;
14.     int _minute;
15.     int _second;
16.
17.     friend Time operator+(const int n, const Time t);
18.     friend Time operator-(const Time t, const int n);
19. public:
20.     Time(int hour = 0, int minute=0, int second=0){
21.         _hour = hour, _minute = minute, _second = second;
22.         setStandard();
23.         setStandardMinus();
24.     }
25.     void setTime(int hour, int minute, int second){
26.         _hour = hour, _minute = minute, _second = second;
27.         setStandard();
28.         setStandardMinus();
29.     }
30.     void setStandard(){
31.         int mod = 0;
32.         if(_second >= 60){
33.             mod = _second / 60;
34.             _second %= 60;
35.         }
36.         _minute += mod;
37.         if(_minute >= 60){
38.             mod = _minute / 60;
39.             _minute %= 60;
40.         }
41.         else{
42.             mod = 0;
43.         }
```

```
44.     _hour += mod;
45.     if(_hour >= 24){
46.         mod = _hour / 24;
47.         _hour %= 24;
48.     }
49. }
50. void setStandardMinus(){
51.     int mod = 0;
52.     if(_second < 0){
53.         mod = (-_second + 59) / 60; //向上取整
54.         _second = _second%60 + 60; //向上取整
55.     }
56.     _minute -= mod;
57.     if(_minute < 0 ){
58.         mod = (-_minute + 59) / 60;
59.         _minute = _minute%60 + 60;
60.     }
61.     else{
62.         mod = 0;
63.     }
64.     _hour -= mod;
65.     if(_hour < 0){
66.         _hour = _hour % 24 + 24;
67.     }
68. }
69. Time& operator++(){
70.     ++_second;
71.     setStandard();
72.     return *this;
73. }
74. Time operator++(int){
75.     Time tmp = *this;
76.     ++_second;
77.     setStandard();
78.     return tmp;
79. }
80.
81. Time& operator--(){
82.     --_second;
83.     setStandardMinus();
84.     return *this;
85. }
86. Time operator--(int){
87.     Time tmp = *this;
```

```

88.     --_second;
89.     setStandardMinus();
90.     return tmp;
91. }
92. void printTime();
93. };
94.
95. void Time::printTime()
96. {
97.     cout<<setfill('0')<<setw(2)<<_hour
98.     <<":"<<setw(2)<<_minute<<":"
99.     <<setw(2)<<_second<<endl;
100. }
101. Time operator+(const int n, const Time t){
102.     Time tmp = t;
103.     tmp._second += n;
104.     tmp.setStandard();
105.     tmp.setStandardMinus();
106.     return tmp;
107. }
108. Time operator-(const Time t, const int n){
109.     Time tmp = t;
110.     tmp._second -= n;
111.     tmp.setStandardMinus();
112.     tmp.setStandard();
113.     return tmp;
114. }

```

时间类:

```
1.  #include <iostream>
2.  #include <iomanip>
3.  using std::cin;
4.  using std::cout;
5.  using std::endl;
6.  using std::setfill;
7.  using std::setw;
8.  //StudybarCommentEnd
9.  class Time{
10. private:
11.     int _hour;
12.     int _minute;
13.     int _second;
14.
15.     friend Time operator+(const int n, const Time t);
16.     friend Time operator-(const Time t, const int n);
17. public:
18.     Time(int hour = 0, int minute=0, int second=0){
19.         _hour = hour, _minute = minute, _second = second;
20.         setStandard();
21.         setStandardMinus();
22.     }
23.     void setTime(int hour, int minute, int second){
24.         _hour = hour, _minute = minute, _second = second;
25.         setStandard();
26.         setStandardMinus();
27.     }
28.     void setStandard(){
29.         int mod = 0;
30.         if(_second >= 60){
31.             mod = _second / 60;
32.             _second %= 60;
33.         }
34.         _minute += mod;
35.         if(_minute >= 60){
36.             mod = _minute / 60;
37.             _minute %= 60;
38.         }
39.         else{
40.             mod = 0;
41.         }
42.         _hour += mod;
43.         if(_hour >= 24){
```

```
44.     mod = _hour / 24;
45.     _hour %= 24;
46. }
47. }
48. void setStandardMinus(){
49.     int mod = 0;
50.     if(_second < 0){
51.         mod = (-_second + 59) / 60; //向上取整
52.         _second = _second%60 + 60; //向上取整
53.     }
54.     _minute -= mod;
55.     if(_minute < 0 ){
56.         mod = (-_minute + 59) / 60;
57.         _minute = _minute%60 + 60;
58.     }
59.     else{
60.         mod = 0;
61.     }
62.     _hour -= mod;
63.     if(_hour < 0){
64.         _hour = _hour % 24 + 24;
65.     }
66. }
67. Time& operator++(){
68.     ++_second;
69.     setStandard();
70.     return *this;
71. }
72. Time operator++(int){
73.     Time tmp = *this;
74.     ++_second;
75.     setStandard();
76.     return tmp;
77. }
78.
79. Time& operator--(){
80.     --_second;
81.     setStandardMinus();
82.     return *this;
83. }
84. Time operator--(int){
85.     Time tmp = *this;
86.     --_second;
87.     setStandardMinus();
```

```

88.     return tmp;
89. }
90. void printTime();
91. };
92. void Time::printTime()
93. {
94.     cout<<setfill('0')<<setw(2)<<_hour
95.     <<":"<<setw(2)<<_minute<<":"
96.     <<setw(2)<<_second<<endl;
97. }
98. Time operator+(const int n, const Time t){
99.     Time tmp = t;
100.    tmp._second += n;
101.    tmp.setStandard();
102.    tmp.setStandardMinus();
103.    return tmp;
104. }
105. Time operator-(const Time t, const int n){
106.     Time tmp = t;
107.     tmp._second -= n;
108.     tmp.setStandardMinus();
109.     tmp.setStandard();
110.     return tmp;
111. }

```

动态数组类:

```

1.     #include <iostream>
2.     #include <cassert>
3.     #include <cstring>
4.     using namespace std;
5.
6.     class Point
7.     {
8.         int x,y;
9.         friend ostream& operator<<(ostream& os, const Point& pt);
10.    public:
11.        //构造函数, 输出 cout<<"\nPoint is called!"; 并完成私有成员的初始化
12.        Point(int x = 0, int y=0){
13.            cout<<"\nPoint is called!";
14.            this->x = x, this->y = y;
15.        }
16.        //析构函数, 输出 cout<<"\n~Point is called!";
17.        ~Point(){

```

```

18.         cout<<"\n~Point is called!";
19.     }
20. };
21. //友元输出函数, 输出    "("<<p.x<<" "<<p.y<<"")";
22. ostream& operator<<(ostream& os, const Point& pt){
23.     os<<"("<<pt.x<<" "<<pt.y<<"")";
24.     return os;
25. }
26. template <typename T>
27. class DynamicArray {
28. private:
29.     T* array; //pointer , 一个T类型的指针
30.     unsigned int mallocSize; //分配空间的大小。
31. public:
32.     //Constructors
33.     // cout<<endl<< "new T["<<this->mallocSize<<"] malloc "<< this->mallocSize
    << "*"<<sizeof(T)<<"]="<<this->mallocSize *sizeof(T)<<" bytes memory in heap";
34.     DynamicArray(){
35.         array = nullptr;
36.         mallocSize = 0;
37.     };
38.     DynamicArray(unsigned length, const T &content=T(0)){
39.         mallocSize = length;
40.         cout<<endl<< "new T["<<this->mallocSize<<"] malloc "<< this->mallocSize <<
    "*"<<sizeof(T)<<"]="<<this->mallocSize *sizeof(T)<<" bytes memory in heap";
41.         array = new T[mallocSize];
42.         for(int i = 0; i< mallocSize; i++){
43.             array[i] = content;
44.         }
45.
46.     }; // mallocSize=length; 设置每个元素的初始内容是 content;
47.     //Copy Constructor
48.     DynamicArray(const DynamicArray<T> & anotherDA ) {
49.         // 拷贝逻辑(深拷贝)
50.         cout<<endl<< "Copy Constructor is called";
51.         mallocSize = anotherDA.mallocSize;
52.         array = new T[mallocSize];
53.         // memcpy(array, anotherDA.array, mallocSize*sizeof(T));
54.         for(int i=0; i<mallocSize; i++){
55.             array[i] = anotherDA.array[i];
56.         }
57.     };
58.     // Destructors

```

```

59.     // cout<<endl<< "delete[] array free "<< this->mallocSize << "*"<<sizeof(T)
    <<"="<<this->mallocSize *sizeof(T)<<" bytes memory in heap";
60.     ~DynamicArray(){
61.         cout<<endl<< "delete[] array free "<< this->mallocSize << "*"<<sizeof(T)<
    <"="<<this->mallocSize *sizeof(T)<<" bytes memory in heap";
62.         delete []array;
63.     };
64.     //return the this->mallocSize
65.     unsigned int capacity() const{
66.         return this->mallocSize;
67.     };
68.     // for the array[i]=someT.
69.     T& operator[](unsigned int i) {
70.         return *(array+i);
71.     };
72.     const T& operator[](unsigned int i) const{
73.         return *(array+i);
74.     };
75.     // 函数内要输出
76.     DynamicArray<T>& operator=(const DynamicArray<T> & anotherDA ) {
77.         cout<<endl<<"operator = is called";
78.         if(this != &anotherDA){
79.             // 赋值拷贝要将原来的对象析构掉
80.             delete []array;
81.             mallocSize = anotherDA.mallocSize;
82.             array = new T[mallocSize];
83.             // memcpy(array, anotherDA.array, mallocSize*sizeof(T));
84.             for(int i=0; i<mallocSize; i++){
85.                 array[i] = anotherDA.array[i];
86.             }
87.         }
88.         return *this;
89.     }
90.     int resize(unsigned int newSize, const T& ValOfNewItems) {
91.         cout<<"\nresize is called";
92.         if(newSize > mallocSize) {
93.             T* tmp = new T[newSize];
94.             for (int i = 0; i < mallocSize; i++) {
95.                 tmp[i] = array[i];
96.             }
97.             for (unsigned int i = mallocSize; i < newSize; i++) {
98.                 tmp[i] = ValOfNewItems;
99.             }
100.            delete[] array;

```



```
101.         array = tmp;
102.         mallocSize = newSize;
103.         return 1;
104.     } else if(newSize < mallocSize) {
105.         T* tmp = new T[newSize];
106.         for (unsigned int i = 0; i < newSize; i++) {
107.             tmp[i] = array[i];
108.         }
109.         delete[] array;
110.         array = tmp;
111.         mallocSize = newSize;
112.         return -1;
113.     } else {
114.         return 0;
115.     }
116. }
117. };
```

大整数类:

```
1.     #include <iostream>
2.     #include <cctype> // isdigit function prototype
3.     #include <cstring> // strlen function prototype
4.     #include <cstdlib>
5.     #include <cmath>
6.     #include <exception>
7.     using namespace std;
8.
9.     class HugeInt
10.    {
11.        friend ostream &operator<< ( ostream &, const HugeInt & );
12.    public:
13.        static const int digits = 30;
14.        HugeInt( long = 0 ); // conversion/default constructor
15.        HugeInt( const char * ); // conversion constructor
16.        // addition operator; HugeInt + HugeInt
17.        HugeInt operator+( const HugeInt & ) const;
18.        HugeInt operator-( const HugeInt & ) const;
19.        HugeInt operator*( const HugeInt & ) const;
20.        HugeInt operator*(int ) const;
21.        HugeInt operator/( const HugeInt & ) const;
22.        // addition operator; HugeInt + int
23.        HugeInt operator+( int ) const;
24.        // addition operator;
25.        // HugeInt + string that represents large integer value
26.        HugeInt operator+( const char * ) const;
27.
28.        int getLength() const;
29.        bool operator>( const HugeInt & ) const;
30.        bool operator<=( const HugeInt & ) const;
31.        bool operator<( const HugeInt & ) const;
32.        bool operator>=( const HugeInt & ) const;
33.        bool operator==( const HugeInt & ) const;
34.        bool operator!=( const HugeInt & ) const;
35.    private:
36.        short integer[ digits ];
37.    }; // end class HugeInt
38.
39.    HugeInt::HugeInt( long num){
40.        for(int i=0; i<digits; i++){
41.            integer[i] = 0;
42.        }
43.        int pos = digits-1;
```

```
44.     while(num>0){
45.         integer[pos--] = num % 10;
46.         num /= 10;
47.     }
48. }
49. HugeInt::HugeInt( const char * num){
50.     // 初始化为0
51.     for(int i=0; i<digits; i++){
52.         integer[i] = 0;
53.     }
54.     int pos = digits-1;
55.     int len = strlen(num);
56.     int posLen = len-1;
57.     while(posLen>=0){
58.         integer[pos--] = (num[posLen--]-'0');
59.     }
60. }
61. HugeInt HugeInt::operator+( const HugeInt & x) const{
62.     HugeInt tmp(*this);
63.     for(int i = digits-1; i>=0; i--){
64.         tmp.integer[i] += x.integer[i];
65.         if(tmp.integer[i]>9){
66.             tmp.integer[i] %= 10;
67.             tmp.integer[i-1] += 1;
68.         }
69.     }
70.     return tmp;
71. }
72. HugeInt HugeInt::operator-( const HugeInt & x) const{
73.     HugeInt tmp(*this);
74.     for(int i = digits-1; i>=0; i--){
75.         tmp.integer[i] -= x.integer[i];
76.         if(tmp.integer[i]<0){
77.             tmp.integer[i] += 10;
78.             tmp.integer[i-1] -= 1;
79.         }
80.     }
81.     return tmp;
82. }
83. HugeInt HugeInt::operator*( const HugeInt & x) const{
84.     HugeInt tmp;
85.     // 先累积
86.     for(int i = digits-1; i>=0; i--){
87.         for(int j = digits-1; j>=0; j--)
```

```
88.     tmp.integer[i+j-digits+1] += integer[j]*x.integer[i];
89. }
90. //进位
91. for(int i = digits-1; i>=0; i--){
92.     if(tmp.integer[i]>9){
93.         tmp.integer[i-1] += tmp.integer[i]/10;
94.         tmp.integer[i] %= 10;
95.     }
96. }
97. return tmp;
98. }
99. HugeInt HugeInt::operator*(int x) const{
100.     HugeInt tmp(x);
101.     return (*this)*tmp;
102. }
103. int getCount(const HugeInt& num, const HugeInt& base){
104.     if(num < base) return 0;
105.     int len_diff = num.getLength() - base.getLength();
106.     int multiplier = std::pow(10, len_diff);
107.
108.     while (true) {
109.         HugeInt base_scaled = base * multiplier;
110.         if (base_scaled <= num) break;
111.
112.         // 幂次过高时需要递减
113.         multiplier = std::pow(10, --len_diff);
114.         if (len_diff < 0) return 0;
115.     }
116.
117.     // 二分查找
118.     int low = 0, high = 9, best = 0;
119.     HugeInt base_scaled = base * multiplier;
120.
121.     while (low <= high) {
122.         int mid = (low + high + 1) / 2; // 修正中点计算
123.         HugeInt temp = base_scaled * mid;
124.
125.         if (temp == num) {
126.             best = mid;
127.             break;
128.         } else if (temp < num) {
129.             best = mid;    // 记录可能的最大值
130.             low = mid + 1;
131.         } else {
```

```
132.         high = mid - 1;
133.     }
134. }
135. // 处理余数部分
136. HugeInt remainder = num - base_scaled * best;
137. int sub_result = getCount(remainder, base);
138. return best * multiplier + sub_result;
139. }
140. HugeInt HugeInt::operator/( const HugeInt & x) const{
141.     if (x == HugeInt()) exit(-1);
142.     if (*this < x) return HugeInt();
143.     return HugeInt(getCount(*this, x));
144. }
145. HugeInt HugeInt::operator+( int x) const{
146.     HugeInt tmp(x);
147.     return *this+tmp;
148. }
149. HugeInt HugeInt::operator+(const char * x) const{
150.     HugeInt tmp(x);
151.     return *this+tmp;
152. }
153. int HugeInt::getLength() const{
154.     int i=0;
155.     for(i=0; i<digits; i++){
156.         if(integer[i]!=0){
157.             break;
158.         }
159.     }
160.     return digits - i;
161. }
162. ostream &operator<<( ostream & os, const HugeInt & x){
163.     int i=0;
164.     for(i=0; i<HugeInt::digits; i++){
165.         if(x.integer[i]!=0){
166.             break;
167.         }
168.     }
169.     if(i!=HugeInt::digits){
170.         for(;i<HugeInt::digits; i++){
171.             os<<x.integer[i];
172.         }
173.     }
174.     else{
175.         os<<0;
```

```
176.     }
177.
178.     return os;
179. }
180. bool HugeInt::operator>( const HugeInt & x) const{
181.     for(int i=0; i<digits; i++){
182.         if(integer[i]>x.integer[i]){
183.             return true;
184.         }
185.         else if(integer[i]<x.integer[i])
186.         {
187.             return false;
188.         }
189.     }
190.     return false;
191. }
192. bool HugeInt::operator<=( const HugeInt & x) const{
193.     return !(*this>x);
194. }
195. bool HugeInt::operator<( const HugeInt & x) const{
196.     for(int i=0; i<digits; i++){
197.         if(integer[i]>x.integer[i]){
198.             return false;
199.         }
200.         else if(integer[i]<x.integer[i])
201.         {
202.             return true;
203.         }
204.     }
205.     return false;
206. }
207. bool HugeInt::operator>=( const HugeInt & x) const{
208.     return !(*this<x);
209. }
210. bool HugeInt::operator==( const HugeInt & x) const{
211.     for(int i=0; i<digits; i++){
212.         if(integer[i]!=x.integer[i]){
213.             return false;
214.         }
215.     }
216.     return true;
217. }
218. bool HugeInt::operator!=( const HugeInt & x) const{
219.     return !(*this==x);}
```

