时间类：

```
#include <iostream>
using namespace std;
class Cmytime{
private:
 int _h, _m, _s;
public:
 Cmytime(int h = 0, int m = 0, int s = 0){
  _h = h, _m = m; _s = s;
 }
 int Set(int h,int m,int s){
  if(h<0 || h>23 || m<0 || m>59 || s<0 || s>59){

   return 0;
  }
  _h = h, _m = m; _s = s;
  return 1;
 }
 void Show(){
  cout<<_h<<":"<<_m<<":"<<_s;
 }
 void  AddOneSecond(){
  _s += 1;
  if(_s == 60){
   _s = 0;
   _m += 1;
  }
  if(_m == 60){
   _m = 0;
   _h += 1;
  }
  if(_h == 24){
   _h = 0;
  }
 }
 int  AddNSeconds(int n){
  _s += n;
  if(_s >= 60){
   int muint = _s / 60;
   _s %=  60;
   _m += muint;
  }
  if(_m >= 60){
   int h_P = _m / 60;
```

```
        _m %= 60;
        _h += h_P;
       }
       if(_h >= 24){
        int day_p = _h / 24;
        _h %= 24;
        return day_p;
       }
       return 0;
     }
    };
    //StudybarCommentBegin
    int main(void) {
        int h,m,s;
        cin>>h>>m>>s;
      Cmytime t1(3,2,1),t2,t3(5);
     t1.Show();
     cout<<"\n";
     t1.Set(h,m,s);
     t1.Show();
     cout<<"\n";
     t2.Show();
     cout<<"\n";
     t3.Show();
        return 0;
    }
    //StudybarCommentEnd
```

时间类综合：

```
    //StudybarCommentBegin
    #include <iostream>
    #include <iomanip>
    using std::cin;
     using std::cout;
     using std::endl;
     using std::setfill;
     using std::setw;
    //StudybarCommentEnd

    class Time{
    private:
     int _hour;
     int _minute;
     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：

```
#include <iostream>
using namespace std;
class GradeBook{
private:
 string _course_name;
 string _instructor_name;
public:
 GradeBook(const string& courseName, const string& structorName){
  _course_name = courseName, _instructor_name = structorName;
 }
 string getInstructorName(){
  return _instructor_name;
 }
 void setInstructorName(const string&  structorName){
  _instructor_name = structorName;
 }
 void displayMessage(){
  cout<<"Welcome to the grade book for"<<endl<<_course_name<<"!"<<endl;
  cout<<"This course is presented by: "<<_instructor_name<<endl;
 }
};
```

Account 类：

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

class Account{
public:
 double _Balance;
 explicit Account(double balance){
  if(balance < 0){
   cout<<"\nError:Initial balance cannot be negative.\n"<<endl;
   balance = 0;
  }
  _Balance = balance;
 }
 void credit(double money){
  _Balance += money;
 }
 void debit(double money){
  if(money <= _Balance){
   _Balance -= money;
  }
```

```
        else
        {cout<<"Debit amount exceeded account balance.\n"<<endl; }
      }
      double getBalance(){
        return _Balance;
      }
    };
```

Invoice 类：

```
    #include <iostream>
    using namespace std;
    class Invoice{
    private:
      string _part_number;
      string _part_description;
      int _quantity;
      int _price_per_item;
    public:
      Invoice(string part_number = "111", string part_description = "None", int q
  uantity = 0, int price_per_item = 0){
        if(quantity<0){
          quantity = 0;
        }
        if(price_per_item<0){
          price_per_item = 0;
        }
        _part_number = part_number, _part_description = part_description;
        _quantity = quantity;
        _price_per_item = price_per_item;
      }
      string getPartNumber(){
        return _part_number;
      }
      string getPartDescription(){
        return _part_description;
      }
      int getQuantity(){
        return _quantity;
      }
      int getPricePerItem(){
        return _price_per_item;
      }
      int getInvoiceAmount(){
        return _quantity*_price_per_item;
```

```
    }
    void setPartNumber(const string& part_number){
     _part_number = part_number;
    }
    void setPartDescription(const string& part_description){
     _part_description  = part_description;
    }
    void setQuantity(const int& quantity){
     if(quantity<0){
      _quantity = 0;
      cout<<"quantity set to 0";
     }
     else{
      _quantity = quantity;
     }
    }
    void setPricePerItem(const int& price_per_item){
     if(price_per_item < 0){
      _price_per_item = 0;
     }
     else{
      _price_per_item = price_per_item;
     }
    }
};
//StudybarCommentBegin
int main()
{
    // create an Invoice object
    Invoice invoice( "12345", "Hammer", 100, 5 );
    // display the invoice data members and calculate the amount
    cout << "Part number: " << invoice.getPartNumber() << endl;
    cout << "Part description: " << invoice.getPartDescription() << endl;
    cout << "Quantity: " << invoice.getQuantity() << endl;
    cout << "Price per item: $" << invoice.getPricePerItem() << endl;
    cout << "Invoice amount: $" << invoice.getInvoiceAmount() << endl;
    // modify the invoice data members
    invoice.setPartNumber( "123456" );
    invoice.setPartDescription( "Saw" );
    invoice.setQuantity( -5 ); // negative quantity, so quantity set to 0
    invoice.setPricePerItem( 10 );
    cout << "\nInvoice data members modified.\n";
    // display the modified invoice data members and calculate new amount
    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：

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

int gcd(int a, int b){
 a = a < 0 ? -a : a;
    b = b < 0 ? -b : b;
 if( b > a){
  int tmp = a;
  a = b;
  b = tmp;
 }
 while (b != 0) {
        int remainder = a % b;
        a = b;
        b = remainder;
    }
    return a;
}
int lcm(int a, int b) {
    if (a == 0 || b == 0) return 0;
    return (a * b) / gcd(a, b);
}
class Fraction{
private:
 int _mole; //分子
 int _deno; //分母

 friend void fswap(Fraction& a, Fraction& b);
 friend void printtest(Fraction& a);
 friend ostream& operator<<(ostream& os, const Fraction& x);
 friend istream& operator>>(istream& in, Fraction& x);
public:
 Fraction(int mole = 1, int deno = 1){
  //处理分子分母负号问题
  if(mole*deno<0){
   mole = -abs(mole);
   deno = abs(deno);
  }
  else if(mole<0 && deno<0){
   mole = abs(mole);
   deno = abs(deno);
```

```
    }
    //处理最简分数
    int gcbNum = gcd(mole, deno);
    mole /= gcbNum;
    deno /= gcbNum;
    //赋值
    _mole = mole;
    _deno = deno;
  }
  Fraction(double x){
    int sign = 1;
    while(int(x)!=x){
      x *= 10;
      sign *= 10;
    }
    int gcbNum = gcd(int(x), sign);
    x /= gcbNum;
    sign /= gcbNum;
    //赋值
    _mole = int(x);
    _deno = sign;
  }
  int getMole(){
    return _mole;
  }
  int getDeno(){
    return _deno;
  }
  void Show(){
    cout<<_mole<<"/"<<_deno;
  }
  double to_double(){
    return _mole*1.0/_deno;
  }

  Fraction operator-(const Fraction& b){
    int deno = lcm(_deno, b._deno);
    int mole = _mole*(deno/_deno) - b._mole*(deno/b._deno);
    return Fraction(mole, deno);
  }
  Fraction operator+(const Fraction& b){
    int deno = lcm(_deno, b._deno);
    int mole = _mole*(deno/_deno) + b._mole*(deno/b._deno);
    return Fraction(mole, deno);
```

```
    }
    Fraction operator*(const Fraction& b){
     return Fraction(_mole*b._mole, _deno*b._deno);
    }
    Fraction operator/(const Fraction& b){
     return Fraction(_mole*b._deno, _deno*b._mole);
    }
    bool operator==(const Fraction& b ){
     return _mole==b._mole && _deno==b._deno;
    }
    bool operator!=(const Fraction& b ){
     return !(*this==b);
    }
    bool operator>(const Fraction& b){
     Fraction c = b;
     return to_double() > c.to_double();
    }
    bool operator<=(const Fraction& b){
     return !(*this > b);
    }
    bool operator<(const Fraction& b){
     Fraction c = b;
     return to_double() < c.to_double();
    }
    bool operator>=(const Fraction& b){
     return !(*this < b);
    }
};
void fswap(Fraction& a, Fraction& b){
 Fraction tmp = a;
 a = b;
 b = tmp;
}
ostream& operator<<(ostream& os, const Fraction& x){
 os<<x._mole<<"/"<<x._deno;
 return os;
}
istream& operator>>(istream& in, Fraction& x) {
    int mole, deno;
    in >> mole >> deno;
    Fraction temp(mole, deno);  // 通过构造函数处理符号和约分
    x._mole = temp.getMole();
    x._deno = temp.getDeno();
    return in;}
```

复数集合类：

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<cmath>
using namespace std;

class Cmycomplex {
private:
    double _real;
    double _imag;
    friend Cmycomplex operator+(const Cmycomplex& cplx, double x);
    friend Cmycomplex operator+(double x, const Cmycomplex& cplx);
    friend std::ostream& operator<<(std::ostream& os, const Cmycomplex& cplx);
    friend std::istream& operator>>(std::istream& is, Cmycomplex& cplx);

public:
    Cmycomplex(double real = 0, double imag = 0) : _real(real), _imag(imag) {}

    double GetReal() const { return _real; }
    double GetImaginary() const { return _imag; }

    // 模计算
    double modulus() const {
        return sqrt(_real * _real + _imag * _imag);
    }

    bool operator==(const Cmycomplex& other) const {
        return _real == other._real && _imag == other._imag;
    }

    void Set(double real, double imag) {
        _real = real;
        _imag = imag;
    }
};
Cmycomplex operator+(const Cmycomplex& cplx, double x){
  return Cmycomplex(cplx._real+x, cplx._imag);
}
Cmycomplex operator+(double x, const Cmycomplex& cplx){
  return Cmycomplex(cplx._real+x, cplx._imag);
}
std::istream& operator>>(std::istream& is, Cmycomplex& cplx) {
    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;
```

```
            for (const auto& e : elements) {
                if (find(other.elements.begin(), other.elements.end(), e) == other.elements.end()) {
                    result.elements.push_back(e);
                }
            }
            return result;
        }
        void Show() const {
            if (elements.empty()) {
                cout << "empty";
                return;
            }
            vector<Cmycomplex> sorted = elements;
            sort(sorted.begin(), sorted.end(), [](const Cmycomplex& a, const Cmycomplex& b) {
                return a.modulus() < b.modulus();
            });
            for (size_t i = 0; i < sorted.size(); ++i) {
                cout << sorted[i] << (i != sorted.size()-1 ? " " : "");
            }
        }
};
```

复数类：

```
#include<iostream>
#include<iomanip>
#include <sstream>
#include <cmath>
using namespace std;

class Cmycomplex{
private:
 double _real;
 double _imag;
 friend Cmycomplex operator+(const Cmycomplex& cplx, double x);
 friend Cmycomplex operator+(double x, const Cmycomplex& cplx);
 friend std::ostream& operator<<(std::ostream& os, const Cmycomplex& cplx);
 friend std::istream& operator>>(std::istream& is, Cmycomplex& cplx);
public:
 Cmycomplex(double real = 0, double imag = 0){
  _real = real;
  _imag = imag;
 }
 void Show(){
  cout <<setiosflags(ios::fixed);
  if(_imag<0){
   cout<<"("<<setprecision(2)<<_real<<setprecision(2)<<_imag<<"i)"<<endl;
  }
  else if(_imag==0){
   cout<<setprecision(2)<<_real<<endl;
  }
  else{
   cout<<"("<<setprecision(2)<<_real<<"+"<<setprecision(2)<<_imag<<"i)"<<end
l;
  }
 }
// void Show(){
//  if(_imag<0){
//   cout<<"("<<_real<<_imag<<"i)"<<endl;
//  }
//  else{
//   cout<<"("<<_real<<"+"<<_imag<<"i)"<<endl;
//  }
// }
 double GetReal(){
  return _real;
 }
```

```
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.     Cmycomplex operator/(const Cmycomplex& x){
61.      double newReal = (_real*x._real+_imag*x._imag)/(x._real*x._real + x._imag*
    x._imag);
62.      double newImag = (_imag*x._real-_real*x._imag)/(x._real*x._real + x._imag*
    x._imag);
63.      return Cmycomplex(newReal, newImag);
64.     }
65.     void Set(double real, double imag){
66.      _real = real;
67.      _imag = imag;
68.     }
69.     bool operator==(const Cmycomplex& x){
70.      return _real==x._real && _imag == x._imag;
71.     }
72.     bool operator!=(const Cmycomplex& x){
73.      return _real!=x._real || _imag != x._imag;
74.     }
75.     Cmycomplex sqrt(){
76.      double newReal = std::sqrt((_real+std::sqrt(_real*_real+_imag*_imag))/2);
77.      double newImag = _imag/std::sqrt(2*_real+2*std::sqrt(_real*_real+_imag*_im
    ag));
78.      return Cmycomplex(newReal, newImag);
79.     }
80.    };
81.    Cmycomplex operator+(const Cmycomplex& cplx, double x){
82.      return Cmycomplex(cplx._real+x, cplx._imag);
```

```
}
Cmycomplex operator+(double x, const Cmycomplex& cplx){
  return Cmycomplex(cplx._real+x, cplx._imag);
}
std::ostream& operator<<(std::ostream& os, const Cmycomplex& cplx) {
 os<<std::fixed<<setprecision(2);
 if(cplx._imag<0){
  os<<cplx._real<<cplx._imag<<"i";
 }
 else if(cplx._imag==0){
  os<<cplx._real;
 }
 else{
  os<<cplx._real<<"+"<<cplx._imag<<"i";
 }
    return os;
}
std::istream& operator>>(std::istream& is, Cmycomplex& cplx) {
    string s;
    cin>>s;
    stringstream ss(s);
    if(s.find('i')!=string::npos){
     //有虚部
     if(s.find('+', 1)!=string::npos||s.find('-', 1)!=string::npos){
      // 复数
       ss>>cplx._real>>cplx._imag;
   ss.get(); // 消耗i
  }
  else{
   //纯虚数
    cplx._real = 0.0;
    ss>>cplx._imag;
    ss.get();// 消耗i
  }
 }
 else{
  // 退化为实数
  cplx._imag = 0.0;
  ss>>cplx._real;
 }
    return is;
}
typedef Cmycomplex ComplexNumber;
```

复数求二次函数：

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

typedef struct {
    double real;
    double imag;
} Complex;
// 复数加法
Complex add(Complex a, Complex b) {
    return (Complex){a.real + b.real, a.imag + b.imag};
}
// 复数减法
Complex sub(Complex a, Complex b) {
    return (Complex){a.real - b.real, a.imag - b.imag};
}
// 复数乘法
Complex mul(Complex a, Complex b) {
    double real = a.real * b.real - a.imag * b.imag;
    double imag = a.real * b.imag + a.imag * b.real;
    return (Complex){real, imag};
}
// 复数除法
Complex div(Complex a, Complex b) {
    double denominator = b.real * b.real + b.imag * b.imag;
    double real = (a.real * b.real + a.imag * b.imag) / denominator;
    double imag = (a.imag * b.real - a.real * b.imag) / denominator;
    return (Complex){real, imag};
}
// 复数平方根
Complex sqrt_complex(Complex z) {
    double r = sqrt(z.real * z.real + z.imag * z.imag);
    double theta = atan2(z.imag, z.real);
    double sqrt_r = sqrt(r);
    double angle = theta / 2;
    return (Complex){sqrt_r * cos(angle), sqrt_r * sin(angle)};
}
// 格式化输出复数
void print_complex(Complex c) {
    if (c.imag >= 0)
        printf("(%.2lf+%.2lfi)\n", c.real, c.imag);
    else
        printf("(%.2lf-%.2lfi)\n", c.real, -c.imag);
```

```
}
int main() {
    Complex a, b, c;
    cin>>a.real>>a.imag;
    cin>>b.real>>b.imag;
 cin>>c.real>>c.imag;
    Complex D = sub(mul(b, b), mul(mul((Complex){4, 0}, a), c));
    Complex sqrtD = sqrt_complex(D);
    Complex root1 = div(add(sub((Complex){0, 0}, b), sqrtD), mul((Complex){2, 0}, a));
    Complex root2 = div(sub(sub((Complex){0, 0}, b), sqrtD), mul((Complex){2, 0}, a));
    if (root2.imag > root1.imag) {
        Complex temp = root1;
        root1 = root2;
        root2 = temp;
    }
    print_complex(root1);
    print_complex(root2);
    return 0;
}
```

长方形类：

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <cstdlib>
using namespace std;

class Rectangle {
private:
  double points_[4][2];
  double length_;
  double width_;

  void SetPoints(const double rect[4][2]) {
    // 调整坐标并验证有效性
    for (int i = 0; i < 4; ++i) {
      // 处理横坐标
      if (rect[i][0] < 0 || rect[i][0] > 20.0) {
        cout << "第" << (i+1) << "个点的横坐标值无效,被置为0" << endl;
        points_[i][0] = 0;
      } else {
        points_[i][0] = rect[i][0];
      }
      // 处理纵坐标
      if (rect[i][1] < 0 || rect[i][1] > 20.0) {
        cout << "第" << (i+1) << "个点的纵坐标值无效,被置为0" << endl;
        points_[i][1] = 0;
      } else {
        points_[i][1] = rect[i][1];
      }
    }
    // 定义点结构并排序
    struct Point {
      double x, y;
      bool operator<(const Point& other) const {
        return (y != other.y) ? (y > other.y) : (x < other.x);
      }
    };
    Point sorted[4];
    for (int i = 0; i < 4; ++i) {
      sorted[i] = {points_[i][0], points_[i][1]};
    }
    sort(sorted, sorted + 4);
    // 验证四边形基本属性
```

```
        const bool top_pair = (sorted[0].y == sorted[1].y);
        const bool bottom_pair = (sorted[2].y == sorted[3].y);
        const bool vertical_edges = (sorted[0].x == sorted[2].x) &&
                                    (sorted[1].x == sorted[3].x);
        const bool horizontal_edges = ((sorted[1].x - sorted[0].x) ==
                                       (sorted[3].x - sorted[2].x));
        const bool vertical_length = ((sorted[0].y - sorted[2].y) ==
                                       (sorted[1].y - sorted[3].y));
        if (!(top_pair && bottom_pair && vertical_edges &&
              horizontal_edges && vertical_length)) {
          cout << "不能构成长方形！" << endl;
          exit(0);
        }
        // 计算长宽
        const double edge_x = sorted[1].x - sorted[0].x;
        const double edge_y = sorted[0].y - sorted[2].y;
        length_ = fmax(edge_x, edge_y);
        width_ = fmin(edge_x, edge_y);
      }
     public:
      explicit Rectangle(const double rect[4][2]) {
        SetPoints(rect);
      }
      double length() const { return length_; }
      double width() const { return width_; }
      double perimeter() const { return 2 * (length_ + width_); }
      double area() const { return length_ * width_; }
      bool square() const { return length_ == width_; }
    };
```

24/12 小时制时间：

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

class Clock24 {
private:
 int _hour,_minute,_second;
public:
 bool IS24;
 Clock24(int h=0,int m=0,int s=0,bool is24=true):IS24(is24),_hour(h%24),_minute(m%60),_second(s%60) {}

 void setStandard(){
  int mod = 0;
  if(_second >= 60){
   mod = _second / 60;
   _second %= 60;
  }
  _minute += mod;
  if(_minute >= 60){
   mod = _minute / 60;
   _minute %= 60;
  }
  else{
   mod = 0;
  }
  _hour += mod;
  if(_hour >= 24){
   mod = _hour / 24;
   _hour %= 24;
  }
 }
 void setStandardMinus(){
  int mod = 0;
  if(_second < 0){
   mod = (-_second + 59)/ 60; //向上取整
   _second = _second%60 + 60; //向上取整
  }
  _minute -= mod;
  if(_minute < 0 ){
   mod = (-_minute + 59) / 60;
   _minute = _minute%60 + 60;
  }
```

```
      else{
       mod = 0;
      }
      _hour -= mod;
      if(_hour < 0){
       _hour =  _hour % 24 + 24;
      }
     }
     Clock24& operator++(){
      ++_second;
      setStandard();
      return *this;
     }
     Clock24 operator++(int){
      Clock24 tmp = *this;
      ++_second;
      setStandard();
      return tmp;
     }
     friend istream& operator>>(istream & in,Clock24 &c);
     friend ostream& operator<<(ostream & out,const Clock24 &c);
     int getSecondsOfDay()const{
       return _second + _minute*60+_hour*3600;
     }
    };
    ostream& operator<<(ostream & out,const Clock24 &c)
    {
     if (c.IS24)
     out<<setfill('0')<<setw(2)<<c._hour<<":"<<setw(2)<<c._minute<<":"<<setw(2)<<c._second;
     else
     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");
     return out;
    }
    istream& operator>>(istream & in,Clock24 &c){
     in>>c._hour>>c._minute>>c._second;
     c.setStandard();
     c.setStandardMinus();
     return in;
    }
```

日期类：

```
//StudybarCommentBegin
#include <iostream>
#include <iomanip>
using std::cin;
 using std::cout;
 using std::endl;
 using std::setfill;
 using std::setw;
//StudybarCommentEnd

class Time{
private:
 int _hour;
 int _minute;
 int _second;

 friend Time operator+(const int n, const Time t);
 friend Time operator-(const Time t, const int n);
public:
 Time(int hour = 0, int minute=0, int second=0){
  _hour = hour, _minute = minute, _second = second;
  setStandard();
  setStandardMinus();
 }
 void setTime(int hour, int minute, int second){
  _hour = hour, _minute = minute, _second = second;
  setStandard();
  setStandardMinus();
 }
 void setStandard(){
  int mod = 0;
  if(_second >= 60){
   mod = _second / 60;
   _second %= 60;
  }
  _minute += mod;
  if(_minute >= 60){
   mod = _minute / 60;
   _minute %= 60;
  }
  else{
   mod = 0;
  }
```

```
    _hour += mod;
    if(_hour >= 24){
     mod = _hour / 24;
     _hour %= 24;
    }
   }
   void setStandardMinus(){
    int mod = 0;
    if(_second < 0){
     mod = (-_second + 59)/ 60; //向上取整
     _second = _second%60 + 60; //向上取整
    }
    _minute -= mod;
    if(_minute < 0 ){
     mod = (-_minute + 59) / 60;
     _minute = _minute%60 + 60;
    }
    else{
     mod = 0;
    }
    _hour -= mod;
    if(_hour < 0){
     _hour =  _hour % 24 + 24;
    }
   }
   Time& operator++(){
    ++_second;
    setStandard();
    return *this;
   }
   Time operator++(int){
    Time tmp = *this;
    ++_second;
    setStandard();
    return tmp;
   }

   Time& operator--(){
    --_second;
    setStandardMinus();
    return *this;
   }
   Time operator--(int){
    Time tmp = *this;
```

```
    --_second;
    setStandardMinus();
    return tmp;
   }
   void printTime();
  };

  void Time::printTime()
  {
   cout<<setfill('0')<<setw(2)<<_hour
    <<":"<<setw(2)<<_minute<<":"
    <<setw(2)<<_second<<endl;
  }
  Time operator+(const int n, const Time t){
   Time tmp = t;
   tmp._second += n;
   tmp.setStandard();
   tmp.setStandardMinus();
   return tmp;
  }
  Time operator-(const Time t, const int n){
   Time tmp = t;
   tmp._second -= n;
   tmp.setStandardMinus();
   tmp.setStandard();
   return tmp;
  }
```

时间类：

```
#include <iostream>
#include <iomanip>
using std::cin;
 using std::cout;
 using std::endl;
 using std::setfill;
 using std::setw;
//StudybarCommentEnd
class Time{
private:
 int _hour;
 int _minute;
 int _second;

 friend Time operator+(const int n, const Time t);
 friend Time operator-(const Time t, const int n);
public:
 Time(int hour = 0, int minute=0, int second=0){
  _hour = hour, _minute = minute, _second = second;
  setStandard();
  setStandardMinus();
 }
 void setTime(int hour, int minute, int second){
  _hour = hour, _minute = minute, _second = second;
  setStandard();
  setStandardMinus();
 }
 void setStandard(){
  int mod = 0;
  if(_second >= 60){
   mod = _second / 60;
   _second %= 60;
  }
  _minute += mod;
  if(_minute >= 60){
   mod = _minute / 60;
   _minute %= 60;
  }
  else{
   mod = 0;
  }
  _hour += mod;
  if(_hour >= 24){
```

```
        mod = _hour / 24;
        _hour %= 24;
       }
      }
      void setStandardMinus(){
       int mod = 0;
       if(_second < 0){
        mod = (-_second + 59)/ 60; //向上取整
        _second = _second%60 + 60; //向上取整
       }
       _minute -= mod;
       if(_minute < 0 ){
        mod = (-_minute + 59) / 60;
        _minute = _minute%60 + 60;
       }
       else{
        mod = 0;
       }
       _hour -= mod;
       if(_hour < 0){
        _hour =  _hour % 24 + 24;
       }
      }
      Time& operator++(){
       ++_second;
       setStandard();
       return *this;
      }
      Time operator++(int){
       Time tmp = *this;
       ++_second;
       setStandard();
       return tmp;
      }

      Time& operator--(){
       --_second;
       setStandardMinus();
       return *this;
      }
      Time operator--(int){
       Time tmp = *this;
       --_second;
       setStandardMinus();
```

```
    return tmp;
   }
   void printTime();
  };
  void Time::printTime()
  {
   cout<<setfill('0')<<setw(2)<<_hour
    <<":"<<setw(2)<<_minute<<":"
    <<setw(2)<<_second<<endl;
  }
  Time operator+(const int n, const Time t){
   Time tmp = t;
   tmp._second += n;
   tmp.setStandard();
   tmp.setStandardMinus();
   return tmp;
  }
  Time operator-(const Time t, const int n){
   Time tmp = t;
   tmp._second -= n;
   tmp.setStandardMinus();
   tmp.setStandard();
   return tmp;
  }
```

动态数组类：

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

class Point
{
    int x,y;
    friend ostream& operator<<(ostream& os, const Point& pt);
public:
    //构造函数，输出 cout<<"\nPoint is called!";  并完成私有成员的初始化
    Point(int x = 0, int y=0){
     cout<<"\nPoint is called!";
     this->x = x, this->y = y;
   }
    //析构函数，输出    cout<<"\n~Point is called!";
    ~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;
```

```
            array = tmp;
            mallocSize = newSize;
            return 1;
    } else if(newSize < mallocSize) {
            T* tmp = new T[newSize];
            for (unsigned int i = 0; i < newSize; i++) {
                tmp[i] = array[i];
            }
            delete[] array;
            array = tmp;
            mallocSize = newSize;
     return -1;
    } else {
     return 0;
    }
  }
  };
```

大整数类：

```
#include <iostream>
#include <cctype> // isdigit function prototype
#include <cstring> // strlen function prototype
#include <cstdlib>
#include <cmath>
#include <exception>
using namespace std;

class HugeInt
{
    friend ostream &operator<<( ostream &, const HugeInt & );
public:
    static const int digits = 30;
    HugeInt( long = 0 ); // conversion/default constructor
    HugeInt( const char * ); // conversion constructor
    // addition operator; HugeInt + HugeInt
    HugeInt operator+( const HugeInt & ) const;
    HugeInt operator-( const HugeInt & ) const;
    HugeInt operator*( const HugeInt & ) const;
    HugeInt operator*(int ) const;
    HugeInt operator/( const HugeInt & ) const;
    // addition operator; HugeInt + int
    HugeInt operator+( int ) const;
    // addition operator;
    // HugeInt + string that represents large integer value
    HugeInt operator+( const char * ) const;

    int getLength() const;
    bool operator>( const HugeInt & ) const;
    bool operator<=( const HugeInt & ) const;
    bool operator<( const HugeInt & ) const;
    bool operator>=( const HugeInt & ) const;
    bool operator==( const HugeInt & ) const;
    bool operator!=( const HugeInt & ) const;
private:
    short integer[ digits ];
}; // end class HugeInt

HugeInt::HugeInt( long num){
 for(int i=0; i<digits; i++){
  integer[i] = 0;
 }
 int pos = digits-1;
```

```
    while(num>0){
     integer[pos--] = num % 10;
     num /= 10;
    }
   }
   HugeInt::HugeInt( const char * num){
    // 初始化为0
    for(int i=0; i<digits; i++){
     integer[i] = 0;
    }
    int pos = digits-1;
    int len = strlen(num);
    int posLen = len-1;
    while(posLen>=0){
     integer[pos--] = (num[posLen--]-'0');
    }
   }
   HugeInt HugeInt::operator+( const HugeInt & x) const{
    HugeInt tmp(*this);
    for(int i = digits-1; i>=0; i--){
     tmp.integer[i] += x.integer[i];
     if(tmp.integer[i]>9){
      tmp.integer[i] %= 10;
      tmp.integer[i-1] += 1;
     }
    }
    return tmp;
   }
   HugeInt HugeInt::operator-( const HugeInt & x) const{
    HugeInt tmp(*this);
    for(int i = digits-1; i>=0; i--){
     tmp.integer[i] -= x.integer[i];
     if(tmp.integer[i]<0){
      tmp.integer[i] += 10;
      tmp.integer[i-1] -= 1;
     }
    }
    return tmp;
   }
   HugeInt HugeInt::operator*( const HugeInt & x) const{
    HugeInt tmp;
    // 先累积
    for(int i = digits-1; i>=0; i--){
     for(int j = digits-1; j>=0; j--)
```

```
      tmp.integer[i+j-digits+1] += integer[j]*x.integer[i];
    }
    //进位
    for(int i = digits-1; i>=0; i--){
     if(tmp.integer[i]>9){
      tmp.integer[i-1] += tmp.integer[i]/10;
      tmp.integer[i] %= 10;
     }
    }
    return tmp;
}
HugeInt HugeInt::operator*(int x) const{
 HugeInt tmp(x);
 return (*this)*tmp;
}
int getCount(const HugeInt& num, const HugeInt& base){
 if(num < base) return 0;
 int len_diff = num.getLength() - base.getLength();
 int multiplier = std::pow(10, len_diff);

 while (true) {
        HugeInt base_scaled = base * multiplier;
        if (base_scaled <= num) break;

        // 幂次过高时需要递减
        multiplier = std::pow(10, --len_diff);
        if (len_diff < 0) return 0;
    }

    // 二分查找
    int low = 0, high = 9, best = 0;
    HugeInt base_scaled = base * multiplier;

    while (low <= high) {
        int mid = (low + high + 1) / 2; // 修正中点计算
        HugeInt temp = base_scaled * mid;

        if (temp == num) {
            best = mid;
            break;
        } else if (temp < num) {
            best = mid;    // 记录可能的最大值
            low = mid + 1;
        } else {
```

```
            high = mid - 1;
        }
    }
    // 处理余数部分
    HugeInt remainder = num - base_scaled * best;
    int sub_result = getCount(remainder, base);
    return best * multiplier + sub_result;
}
HugeInt HugeInt::operator/( const HugeInt & x) const{
 if (x == HugeInt()) exit(-1);
    if (*this < x) return HugeInt();
    return HugeInt(getCount(*this, x));
}
HugeInt HugeInt::operator+( int x) const{
 HugeInt tmp(x);
 return *this+tmp;
}
HugeInt HugeInt::operator+(const char *  x) const{
 HugeInt tmp(x);
 return *this+tmp;
}
int HugeInt::getLength() const{
 int i=0;
 for(i=0; i<digits; i++){
  if(integer[i]!=0){
   break;
  }
 }
 return digits - i;
}
ostream &operator<<( ostream & os, const HugeInt & x){
 int i=0;
 for(i=0; i<HugeInt::digits; i++){
  if(x.integer[i]!=0){
   break;
  }
 }
 if(i!=HugeInt::digits){
  for(;i<HugeInt::digits; i++){
   os<<x.integer[i];
  }
 }
 else{
  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);}
```