一．m=0 n=8
二．声明为静态成员，实现同一类不同对象之间的数据共享。
三．解决同名成员的唯一标识问题
四．先调用基类构造函数，再初始化派生类中的新增成员，最后执行派生类的构造函数。
五．

```
1. #include<iostream.h>
       using namespace std;
2. class teacher;
3. class student
4. {
     char *name;
5. public:
6. student(char *s){name=s;}
     friend void print(student &a,teacher &b) ;
7. };
8. class teacher
9. {
     char *name;
10. public:
11. teacher(char *s){name=s;}
         friend void print(student &a,teacher &b) ;
12. };
13. void print(student &a,teacher &b)
14. { cout<<"the student is:"<<a.name<<endl;
15. cout<<"the teacher is:"<<b.name<<endl;
16. }
17. void main()
18. { student s("Bill Gates");
19. teacher t("Beckham");
20. print(s,t);
21. }
```

六．

```
1. void fun(vector < vector<int>> vec)
  { int row = vec.size();
     for (int i = 0; i < row; i++)
          int col = vec[i].size();
          for (int j = 0;j < col; j++)
          cout << vec[i][j] << endl;
     return;
}
```

七.

```
BaseB Constructor
BaseA Constructor
DerivedC Constructor
X=8
X=39
X=16
```

八.

```
void mergeArray(float A[], int lenA, float B[], int lenB, float C[]) {
    for (int i = 0; i < lenB; i++) {
        C[i] = B[i];
    }
    for (int i = 0; i < lenA; i++) {
        C[i+lenB] = A[i];
    }
    for (int i = 0; i < lenA+lenB -1; i++) {
        for (int j = 0; j < lenA+lenB -1 - i ; j++) {
            if (C[j] > C[j + 1]) {
                float t = C[j];
                C[j] = C[j + 1];
                C[j + 1] = t;

            }
        }
    }
}
int main() {
    float A[5] = { 5.4, 6.7, 7.7, 9.2, 10 }; //升序数组 1
    float B[4] = { 3.4, 6.5, 8.7, 9.9 }; //升序数组 2
    float* C = new float[9];
    mergeArray(A, 5, B, 4, C);
    for (int i = 0; i < 9; i++) {
        cout << C[i] << "  ";
    }
}
```

九.

```
class Fraction {
private:
    int numerator, denominator;
```

```
public:
    Fraction(int numerator1, int denominator1) :numerator(numerator1),
denominator(denominator1) {};
    Fraction operator+(Fraction& a);
    Fraction operator*(Fraction& a);
    friend ostream& operator<<(ostream& c, const Fraction f);
};
Fraction Fraction::operator+(Fraction& a) {
    int n1 = numerator;
    n1 = a.denominator * n1;
    int d = denominator;
    int n2 = d * a.numerator;
    int n3 = n2 + n1;
    d = a.denominator * d;
    int tmp = 1;
    for (int i = 2; i <= n3; i++) {
        if (n3 % i == 0 && d % i == 0) {
            tmp = i;
        }
    }
    n3 = n3 / tmp;
    d = d / tmp;
    return Fraction(n3, d);
}
Fraction Fraction::operator*(Fraction& a) {
    int n = numerator * a.numerator;
    int d = denominator * a.denominator;
    int tmp = 1;
    for (int i = 2; i <= n ; i++) {
        if (n % i == 0 && d % i == 0) {
            tmp = i;
        }
    }
    n = n / tmp;
    d = d / tmp;
    return Fraction(n, d);
}
ostream& operator<<(ostream& c, const Fraction f) {
    cout << f.numerator << "/" << f.denominator;
    return cout;
}
int main() {
    Fraction f1(3, 4);
    Fraction f2(2, 3);
```

```
    Fraction f5(1, 2);
    Fraction f6(1, 6);
    Fraction f3 = f1 + f2;
    cout << f3 << endl;
    Fraction f4= f1 * f2;
    cout << f4 << endl;
    Fraction f7 = f5 + f6;
    cout << f7 << endl;
}
```

2019clock 题目

```
#include<iostream>
using namespace std;
class Clock {
private:
int hour, minute, second;
public:
Clock();
Clock(int h, int m, int s);
Clock operator+(const Clock& c)const;
bool operator<(const Clock& c)const;
Clock& operator++();
Clock operator++(int);
void show();
};
Clock::Clock() :hour(0), minute(0), second(0) {
}
Clock::Clock(int h,int m,int s):hour((h + (m / 60)) % 24), minute((m + (s / 60)) % 60),
 second(s % 60) {};
//请补充 Clock 的定义
Clock Clock::operator+(const Clock& c)const {
int htmp = c.hour + hour;
int mtmp = c.minute + minute;
int stmp = c.second + second;
Clock ck((htmp + (mtmp / 60)) % 24, (mtmp + (stmp / 60)) % 60, stmp % 60);
return ck;
}
bool Clock::operator<(const Clock& c)const{
if (hour == c.hour) {
if (minute == c.minute) {
if (second == c.second) {
return 0;
```

```
}
return minute < c.minute;
}
return second < c.second;
}
return hour < c.hour;
}
Clock& Clock::operator++(){//前置++返回引用，主要原因是为了减少内存花销，不用另外创建对象，操作也是对本身的操作
second = (second + 1) % 60;
minute = (minute + second / 60) % 60;
hour = (hour + minute / 60) % 24;
return *this;
};
Clock Clock::operator++(int){//后置++返回拷贝,主要原因是为了实现延时的机制，返回的只是一个拷贝
Clock c = *this;
++(*this);
return c;
}
void Clock::show() {
cout << hour << " " << minute << " " << second << endl;
}
void main() {

Clock c1(2,20, 59);
Clock c2(20, 54, 39);
(c1 + c2).show();
bool b = c1 < c2;
cout << b;
(++c1).show();
c1.show();
(c1++).show();
c1.show();
int m = 8,n = 8,a = 8,c = 8;
n = a >= c;
(m = a > c) && (n = a >= c);
cout << m << " " << n << endl;
}
排序
void dataprocess(int a[], int n){
for (int i = 0; i < n; i++) {
for (int j = i; j < n; j++) {
if (a[i] < a[j]) {
```

```
int tmp = a[i];
a[i] = a[j];
a[j] = tmp;
}
}
}
}
```