1.4

```
//complexnumber.h
#ifndef GUARD_complexnumber_h
#define GUARD_complexnumber_h
#include <iostream>

class complexnumber {
public:
    complexnumber() : real(0), imaginary(0) {}
    complexnumber(double re) : real(re), imaginary(0) {}
    complexnumber(double re, double im) : real(re), imaginary(im) {}
    complexnumber(const complexnumber& rhs) : real(rhs.real),
imaginary(rhs.imaginary) {}
    complexnumber& operator=(const complexnumber& rhs);
    complexnumber& operator+=(const complexnumber& rhs);
    complexnumber& operator-=(const complexnumber& rhs);
    complexnumber& operator*=(const complexnumber& rhs);
    complexnumber& operator/=(const complexnumber& rhs);
    friend std::istream& operator>>(std::istream& is, complexnumber& rhs);
    friend std::ostream& operator<<(std::ostream& os, const complexnumber& rhs);
private:
    double real;
    double imaginary;
};

complexnumber operator+(const complexnumber& lhs, const complexnumber& rhs);
complexnumber operator-(const complexnumber& lhs, const complexnumber& rhs);
complexnumber operator*(const complexnumber& lhs, const complexnumber& rhs);
complexnumber operator/(const complexnumber& lhs, const complexnumber& rhs);

#endif
```

```
//complexnumber.cpp
#include "complexnumber.h"
#include <cmath>

complexnumber& complexnumber::operator=(const complexnumber& rhs)
{
    real = rhs.real;
    imaginary = rhs.imaginary;
    return *this;
}

complexnumber& complexnumber::operator+=(const complexnumber& rhs) {
    real += rhs.real;
    imaginary += rhs.imaginary;
    return *this;
}

complexnumber& complexnumber::operator-=(const complexnumber& rhs) {
    real -= rhs.real;
```

```
    imaginary -= rhs.imaginary;
    return *this;
}

complexnumber& complexnumber::operator*=(const complexnumber& rhs) {
    real = real * rhs.real - imaginary * rhs.imaginary;
    imaginary = real * rhs.imaginary + imaginary * rhs.real;
    return *this;
}

complexnumber& complexnumber::operator/=(const complexnumber& rhs) {
    double denominator = rhs.real * rhs.real + rhs.imaginary * rhs.imaginary;
    real = (real * rhs.real + imaginary * rhs.imaginary) / denominator;
    imaginary = (imaginary * rhs.real - real * rhs.imaginary) / denominator;
    return *this;
}

std::ostream& operator<<(std::ostream& os, const complexnumber& rhs) {
    os << rhs.real << ((rhs.imaginary >= 0) ? "+" : "-") << fabs(rhs.imaginary)
<< "i";
    return os;
}

std::istream& operator>>(std::istream& is, complexnumber& rhs) {
    is >> rhs.real >> rhs.imaginary;
    return is;
}

complexnumber operator+(const complexnumber& lhs, const complexnumber& rhs) {
    complexnumber temp = lhs;
    temp += rhs;
    return temp;
}

complexnumber operator-(const complexnumber& lhs, const complexnumber& rhs) {
    complexnumber temp = lhs;
    temp -= rhs;
    return temp;
}

complexnumber operator*(const complexnumber& lhs, const complexnumber& rhs) {
    complexnumber temp = lhs;
    temp *= rhs;
    return temp;
}

complexnumber operator/(const complexnumber& lhs, const complexnumber& rhs) {
    complexnumber temp = lhs;
    temp /= rhs;
    return temp;
}
```

1.11

(1)

```
void d(int x[], int n) {
    int i = 0; count++;
    do {
        x[i] += 2; i += 2; count += 2;
        count++;
    } while (i <= n);
    i = 0; count++;
    while (i <= (n / 2)) {
        count++;
        x[i] += x[i + 1]; i++; count += 2;
    }
    count++;
}
```

(2)

```
void d(int x[], int n) {
    int i = 0;
    do {
        i += 2; count += 3;
    } while (i <= n);
    i = 0;
    while (i <= (n / 2)) {
        i++; count += 3;
    }
    count += 3;
}
```

(3)

当n为偶数时

$$count = 3 \times \frac{n+2}{2} + 3 \times (\frac{n}{2} + 1) + 3 = 3n + 9$$

当n为奇数时

$$count = 3 \times \frac{n+1}{2} + 3 \times \frac{n+1}{2} + 3 = 3n + 6$$

(4)

当n为偶数时

行号	程序语句	一次执行所需程序步数	执行次数	程序步数
1	void d(int x[], int n) {	0	1	0
2	int i = 0;	1	1	1
3	do {	0	(n+2)/2	0
4	x[i] += 2;	1	(n+2)/2	(n+2)/2
5	i += 2;	1	(n+2)/2	(n+2)/2
6	} while (i <= n);	1	(n+2)/2	(n+2)/2
7	i = 0;	1	1	1
8	while (i <= (n / 2)) {	1	n/2+2	n/2+2
9	x[i] += x[i + 1];	1	n/2+1	n/2+1
10	i++;	1	n/2+1	n/2+1
11	}	0	n/2+1	0
12	}	0	1	0
total				3n+9

当n为奇数时

行号	程序语句	一次执行所需程序步数	执行次数	程序步数
1	void d(int x[], int n) {	0	1	0
2	int i = 0;	1	1	1
3	do {	0	(n+1)/2	0
4	x[i] += 2;	1	(n+1)/2	(n+1)/2
5	i += 2;	1	(n+1)/2	(n+1)/2
6	} while (i <= n);	1	(n+1)/2	(n+1)/2
7	i = 0;	1	1	1
8	while (i <= (n / 2)) {	1	(n+1)/2+1	(n+1)/2+1
9	x[i] += x[i + 1];	1	(n+1)/2	(n+1)/2
10	i++;	1	(n+1)/2	(n+1)/2
11	}	0	(n+1)/2	0
12	}	0	1	0
total				3n+6