---- complex.h ----------

```
#include <iostream>
using std::ostream;

class Complex {
  double real;
  double imag;
public:
  Complex(double r, double i);
  void print(ostream &out) const;
};
ostream &operator<<(ostream &out,
                    const Complex &c);
```

---- complex.cc ----------

```
Complex(double r, double i) : real(r), imag(i) { }

void Complex::print(ostream &out) const
{
  out << real << " + " << imag << "i";
}
ostream &operator<<(ostream &out,const
Complex &c)
{
  c.print(out);
  return out;
}
```

---- main.cc ----------

```
#include <iostream>
#include "complex.h"
using std::cout;
using std::endl;

int main()
{
  Complex c(3,2);
  cout << "complex is : " << c << endl;
  return 0;
}
complex is : 3 + 2i
```

```
#include <set>
#include <string>
using std::string;

typedef string VarName;
typedef double Coeff;

class EqElm {
  VarName name;
  Coeff coeff;
public:
  EqElm(_____ Coeff _) _____;

  EqElm(_____ Coeff _, ______ VarName _) _____;

  EqElm _ operator+=(_____ EqElm _) _____;
};

EqElm _ operator+(_____ EqElm _,
                  _____ EqElm _) _____;


class LinearEq {
  set<EqElm> elements;
public:
  LinearEq() _____;
  LinearEq(_____ EqElm _) _____;

  set<VarName> _ var_names() _____;

  LinearEq _ operator+=(_____ EqElm _) _____;
  LinearEq _ operator*=(_____ Coeff _) _____;
  LinearEq _ operator* (_____ Coeff _) _____;
};

LinearEq _ operator*(_____ Coeff   _,
                     _____ LinearEq _) _____;

LinearEq _ operator+(_____ LinearEq _,
                     _____ LinearEq _) _____;
```

```
#ifndef POINT_H_
#define POINT_H_

template<class Coordinate>
class Point
{
  Coordinate x, y;

public:
  Point(_____ Coordinate _, _____ Coordinate _) ___;

  double _ radius() _____;
  double _ distance(_____ Point _) _____;
  Point  _ middle(_____ Point _) _____;
  double _ arg() _____;

  Point _ operator+(_____ Point _) _____;
  Point _ operator+=(_____ Point _) _____;

   Point _ operator*(_____ double _) _____;
   Point _ operator*=(_____ double _) _____;
};

template<class Coordinate>
Point<Coordinate> _ operator*(_____ double _,
                  _____ Point<Coordinate> _) _____;

#endif
```

```
class Complex {
  double real, imag;
public:
  Complex(double r, double i) : real(r), imag(i) { }
  Complex &operator+=(const Complex &c);
};

Complex operator+(double real, const
Complex &c)
{
  Complex tmp(real, 0);
  tmp += c;
  return tmp;
}

Complex operator+(const Complex &c1,
                  const Complex &c2)
{
  Complex tmp = c1;
  tmp += c2;
  return tmp;
}

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

```
class Complex {
  double real, imag;

public:
  Complex(double r) : real(r), imag(0) { }
  Complex(double r, double i) : real(r), imag(i) {};

  Complex &operator+=(const Complex &c);
};
```

```
Complex operator+(const Complex &c1,
                  const Complex &c2)
{
  Complex tmp = c1;
  tmp += c2;
  return tmp;
}

template<class T>
class Array {
  T *array;
  int size;

  void resize(int new_size);
public:
  Array();
  T &operator[ ] (int k);
  T operator[ ] (int k) const;
};
```

```
template<class T>
Array<T>::Array()
{
  size = array = 0;
}

template<class T>
void Array<T>::resize(int new_size)
{
  T *tmp = new T [new_size];
  for(int i=0; i<size; i++)  tmp[ i ] = array[ i ];

  delete [ ] array;
  size = new_size;
  array = tmp;
}
```

```
template<class T>
T &Array<T>::operator[ ] (int k)
{
  if(k < 0) exit(1);
  if(k >= size) resize(k+1);
  return array[k];
}

template<class T>
T Array<T>::operator[ ] (int k) const
{
  if(k < 0 || k >= size) exit(1);
  return array[k];
}
```

```
int main()
{
  Array<int> arr;

  arr[3] = 50;
  arr[200] = 150;
  arr[10000] = 200;

  func(arr);

  return 0;
}

void func(const Array<int> &a)
{
  for(int i = 0; i<=1000; i++) cout << a[ i ];
}
```