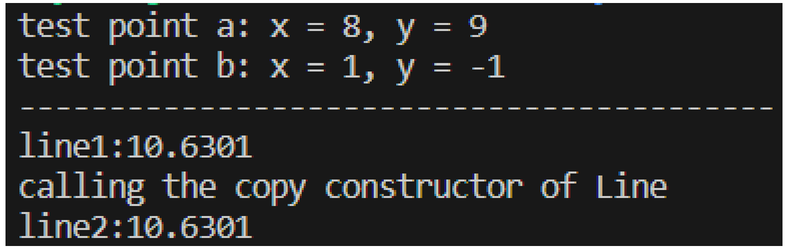
3 Exercises

1. The declarations of Point class and Line class are as follows:

```
class Point {
private:
    double x, y;

 public:
    Point(double newX, double newY) ;

    double getX() const;
    double getY() const;

};
```

```
class Line
{
private:
    Point p1, p2;
    double distance;

public:
     Line(Point xp1, Point xp2);
     Line(Line& q);
    double getDistance() const;
};
```

Complete the member functions of the two classes. Write a program to test the classes.



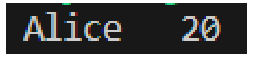
```
test point a: x = 8, y = 9
test point b: x = 1, y = -1
------------------------------------------
line1:10.6301
calling the copy constructor of Line
line2:10.6301
```

2. A template class named **Pair** is defined as follows. Please implement the overloading **operator<** which compares the value of the key, if this->key is smaller than that of p.key, return true. Then define a friend function to overload **<< operator** which displays the Pair's data members. At last, run the program. The output sample is as follows:

```
#include <iostream>
#include <string>
using namespace std;
template <class T1,class T2>
class Pair
{
public:
   T1 key;
   T2 value;
   Pair(T1 k,T2 v):key(k),value(v) { };
   bool operator < (const Pair<T1,T2> & p) const;

};
```

```
int main()
{
   Pair<string,int> one("Tom",19);
   Pair<string,int> two("Alice",20);

   if(one < two)
      cout << one;
   else
      cout << two;

   return 0;
}
```

Output:



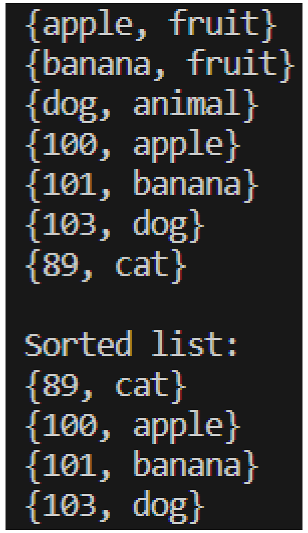
```
Alice   20
```

3. There is a definition of a template class **Dictionary**. Please write a template partial specialization for Dictionary class whose **Key** is specified to be **int,** and add a member function named sort() which sorts the elements in dictionary in ascending order. At last, run the program. The output sample is as follows:

```
template <class Key, class Value>
class Dictionary {
  Key* keys;
  Value* values;
  int size;
  int max_size;
public:
  Dictionary(int initial_size) :  size(0) {
    max_size = 1;
    while (initial_size >= max_size)
      max_size *= 2;
    keys = new Key[max_size];
    values = new Value[max_size];
  }
  void add(Key key, Value value) {
    Key* tmpKey;
    Value* tmpVal;
    if (size + 1 >= max_size) {
      max_size *= 2;
      tmpKey = new Key [max_size];
      tmpVal = new Value [max_size];
      for (int i = 0; i < size; i++) {
        tmpKey[i] = keys[i];
        tmpVal[i] = values[i];
      }
      tmpKey[size] = key;
      tmpVal[size] = value;
      delete[] keys;
      delete[] values;
      keys = tmpKey;
      values = tmpVal;
    }
    else {
      keys[size] = key;
      values[size] = value;
    }
    size++;
  }

  void print() {
    for (int i = 0; i < size; i++)
      cout << "{" << keys[i] << ", " << values[i] << "}" << endl;
  }

~Dictionary()
  {
      delete[] keys;
      delete[] values;
  }

};
```

```
int main()
{
    Dictionary<const char*, const char*> dict(10);
    dict.print();
    dict.add("apple", "fruit");
    dict.add("banana", "fruit");
    dict.add("dog", "animal");
    dict.print();

    Dictionary<int, const char*> dict_specialized(10);
    dict_specialized.print();
    dict_specialized.add(100, "apple");
    dict_specialized.add(101, "banana");
    dict_specialized.add(103, "dog");
    dict_specialized.add(89, "cat");
    dict_specialized.print();
    dict_specialized.sort();
    cout << endl << "Sorted list:" << endl;
    dict_specialized.print();

   return 0;
}
```

Output:



```
{apple, fruit}
{banana, fruit}
{dog, animal}
{100, apple}
{101, banana}
{103, dog}
{89, cat}

Sorted list:
{89, cat}
{100, apple}
{101, banana}
{103, dog}
```