COMP 151: Object Oriented Programming
Spring Semester 2009
Final Exam
Instructor: Chi Keung Tang
Thursday, May 21, 2009
4:30 – 7:30pm
HALL

This is a **CLOSED-BOOK-CLOSED-NOTES** exam consisting of eight (8) problems. Follow the instructions carefully. Please write legibly in the space provided. Keep the exam booklet stapled.

K E Y

Problem	Points	your score
1 MULTIPLE CHOICE	10	
2 OBJECT CONSTRUCTION AND DESTRUCTION	13	
3 INTERFACE VS IMPLEMENTATION	10	
4 ABSTRACT BASE CLASS	15	
5 STATIC CLASS	13	
6 FUNCTION OBJECTS	8	
7 CONTAINER CLASS	18	
8 STL	13	
Total	100	

1 Multiple Choice

Select the answer by drawing a circle around the letter. (1 point for each correct answer; 0 point for each empty or incorrect answer)

1. You can use private inheritance
   (a) to be able to inherit and thus access private variables in the base class
   (b) **to provide a new interface, thus hiding the interface of the base class**
   (c) to make the code more efficient as all references are local
   (d) all of the above
2. Template code can be precompiled like regular code can
   (a) True
   (b) **False**
3. A pure virtual function exists if and only if the pertinent class is an abstract base class
   (a) **True**
   (b) False
4. Inheritance usually represents
   (a) aggregation relationship
   (b) has-a relationship
   (c) **is-a relationship**
   (d) association relationship
5. Code reuse in object-oriented programming means
   (a) Copy/Paste code from a previous project/student.
   (b) Using code found on the web, with slight modifications.
   (c) Using code from a book, with slight modifications.
   (d) **Reusing full class definitions without any modification.**
6. Slicing in object-oriented programming is
   (a) what happens when you order pizza
   (b) **what happens when you assign an object of a derived type to a variable of the base class**
   (c) what happens when you assign an object of a base type to a variable of the derived class.
   (d) none of the above.
7. To get true polymorphism in C++ you need
   (a) object and virtual functions
   (b) **pointers to objects and virtual functions**
   (c) pointers to objects but not virtual functions
   (d) none of the above
8. A pure virtual function

(a) a method defined with body of $= 0$

(b) a way to get polymorphism in C++

(c) may make the code less efficient in run-time.

(d) **all of the above**

9. An abstract base class is a class defined

(a) as `class ClassName { ... } = 0;`

(b) **with at least one pure virtual function**

(c) with at least one member variable declared pure virtual

(d) all of the above

10. Protected member of a base class becomes _____ when inherited via public inheritance.

(a) Public

(b) **Protected**

(c) Private

(d) all of the above

2 Object Construction and Destruction

Write the output of the following short programs on page 7 of the exam.

```
//---------------------program 1 : begin-----------------------
#include <iostream>

class Base {
public:
  Base (): x(0), y(0) { cout << "Base" << endl; }
  Base (int i): x(10), y(10)
    { cout << "Base" << i << " " << x << " " << y << endl; }
  ~Base () { cout << "Destruct Base" << endl; }
private:
  int x,y;
};

class Der : public Base {
public:
  Der (): x(1.0), y(1.0) { cout << "Der" << endl; }
  Der (int i) : x(20.0), y(20.0), Base (i)
    { cout << "Der"  << i << " " << x << " " << y << endl; }
  ~Der () { cout << "Destruct Der" << endl; }
private:
  float x,y;
};

int main () {
  Base a;
  Der d(2);
  return 0;
}
//---------------------program 1 : end-------------------------


//---------------------program 2 : begin-----------------------
#include <iostream>
using namespace std;

class C {
public:
  C() : i(0) { cout << i << endl; }
  C(int in_i) : i(in_i) { cout << i << endl; }
  ~C() { cout << i << endl; }
  void iSet (int x) { i = x; }
private:
  int i;
};

int main() {
```

```
  C c1, c2(7);

  c1.iSet(5);

  {
    static C c3;
    c3.iSet(4);
  }

  C *c4 = new C(2);
  delete c4;
  return 0;

}
//----------------------program 2 : end ------------------------

//----------------------program 3 : begin-----------------------
#include <iostream>

class A {
public:
  int f() { return 1; }
  virtual int g() { return 2; }
};

class B : public A {
public:
  int f() { return 3; }
  virtual int g() { return 4; }
};

class C : public B {
public:
  virtual int g() { return 5; }
};

int main() {
  A *pa;
  A a;
  B b;
  C c;

  pa = &a; cout << pa->f() << endl; cout << pa->g() << endl;
  pa = &b; cout << pa->f() + pa->g() << endl;
  pa = &c; cout << pa->f() << endl; cout << pa->g() << endl;
  cout << c.f() << endl;
  return 0;
}
//----------------------program 3 : end------------------------
```

```
//---------------------program 4 : begin----------------------
#include <iostream>
using namespace std;

class A{
protected:
  int a;
public:
  A (int x=1) { a = x; }
  int f() { a += 3; return a; }
  virtual int g() { a += 4; return a; }
  int h() { f(); return a; }
  int j() { g(); return a; }
};

class B : public A {
private:
  int b;
public:
  B (int y=5) { b = y; }
  int f() { b += 7; return b; }
  int g() { a += 5; return a; }
};

int main()
{
  A obj1;
  B obj2;

  cout << obj1.f() << endl;
  cout << obj1.h() << endl;
  cout << obj1.g() << endl;
  cout << obj2.f() << endl;
  cout << obj2.h() << endl;
  cout << obj2.g() << endl;
}
//---------------------program 4 : end-------------------------
```

Write your answer on the next page.

Write your answer here.

```
Base
Base2 10 10
Der2 20 20
Destruct Der
Destruct Base
Destruct Base
```

```
0
7
0
2
2
7
5
4
```

```
1
2
5
1
5
3
```

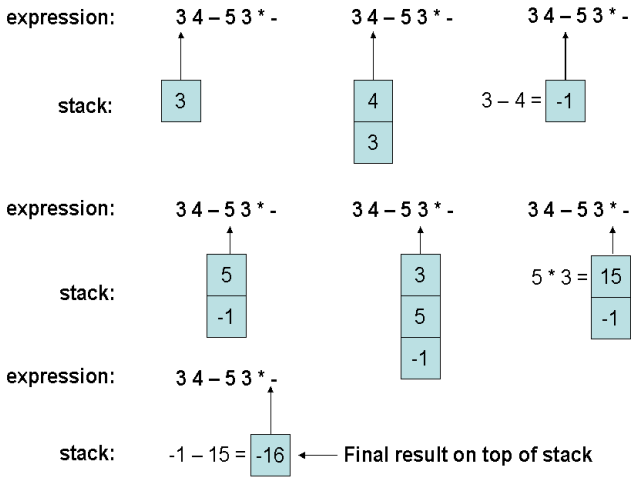
```
4
7
11
12
4
9
```

3 Interface vs Implementation

Write a program to evaluate expressions in *Reverse Polish Notation (RPN)* or *postfix* notation, using the `stack` class and other classes provided for you. The evaluation rule is:

1. Evaluate the given expression from left to right.
2. At each occurrence of an operator, apply it to the two operands to the immediate left and replace the sequence of two operands and one operator by the resulting value.

An example is shown in the figure below.



To simplify the problem, we make use of the following `Token` class and the associated facilities. A tokenizer is a program that reads the input, and tokenizes the input string into a set of tokens which are of the following types:

```
enum tokenType { operandToken, operatorToken, eolToken, eofToken, badToken };
```

Each token will be one of the enumerated types. An operand will be an `operandToken`; an operator will be an `operatorToken`. The end of a line and end of the input file will be signaled by the special `eolToken` and `eofToken`. Finally, if the tokenizer cannot interpret the input, it will mark it as a `badToken`, leaving it to the client to respond appropriately.
The operators are classified using the following declaration:

```
enum operatorType { none, add, subtract, multiply, divide };
```

The `none` type is used if the client mistakenly requests the operator type for a token that's not an operator.
The code below contains the class definition that defines the public interface for the `Token` class:

```
#ifndef TOKEN_H
#define TOKEN_H

class Token {
public:
  tokenType     nextToken();   // get next token from the input stream
  operatorType  getOperator(); // return operator
  double        getOperand();  // return operand

// other declarations...
};

#endif
```

The `Stack` class provided is a class template:

```
#ifndef STACK_H
#define STACK_H

template <class ElementType>
class Stack {
public:
  Stack ();
  void push (ElementType e);
  ElementType pop();     // pop an element
  ElementType top();     // get the top element; stack does not change
  bool isEmpty();        // check stack emptiness
  void clear();          // clear stack
private:
  // private data structures...
};

#endif
```

You must use the following skeleton and complete the program. Use the functions provided to keep your code simple – writing complex and buggy codes are not recommended.

```
#include "utilities.h" // enumerated types definitions
#include "stack.h"     // Stack class
#include "token.h"     // token class

int main()
{
  Token t;
  Stack <double> s;
  double op1, op2;
  bool done(false);

  while (!done) {
    switch (t.nextToken()) {
      case operandToken:









      case operatorToken:
        // op2 is the top of the stack, op1 is the next value down
        // first, have to make sure there are two items to pop
```

```
      case eofToken:
        done = true;   // break intentionally omitted here
      case eolToken:   // if there's something in stack, display it
        if (!s.isEmpty())
          fprintf (stdout, "--> %f\n",  s.pop());
          s.clear();  // clear stack for the next line
        break;
      case badToken:
        fprintf (stderr, "Input error!\n");
        break;
    }
  }
  return 0;
```

Solution:

```
#include "utilities.h" // enumerated types definitions
#include "stack.h"     // Stack class
#include "token.h"     // token class

int main()
{
  Token t;
  Stack <double> s;
  double op1, op2;
  bool done(false);

  while (!done) {
    switch (t.nextToken()) {
      case operandToken:
        s.push(t.getOperand());
        break;
      case operatorToken:
        // op2 is the top of the stack, op1 is the next value down
        // first, have to make sure there are two items to pop
        if ( (s.isEmpty()) || (op2 = s.pop() && s.isEmpty() ))
          fprintf (stderr, "Not enough operands for operator!\n");
        else {  // get op1, then apply appropriate operator
          op1 = s.pop();
          switch (t.getOperator()) {
            case add:      s.push (op1 + op2); break;
            case subtract: s.push (op1 - op2); break;
            case multiply: s.push (op1 * op2); break;
            case divide:
              if (op2 == 0)
                fprintf (stderr, "Division by zero!\n");
              else
                s.push (op1 / op2);
              break;
```

```
            }
          }
          break;
        case eofToken:
          done = true;   // break intentionally omitted here
        case eolToken:   // if there's something in stack, display it
          if (!s.isEmpty())
            fprintf (stdout, "--> %f\n",  s.pop());
          // s.clear();  // clear stack for the next line
          break;
        case badToken:
          fprintf (stderr, "Input error!\n");
          break;
      }
    }
    return 0;
}
```

4 Abstract Base Class

Refer to the following main program:

```
//------------------area.cpp : begin-----------------------------------
#include <stdio.h>
#include "polygon.h"
#include "circle.h"
#include "square.h"

int main() {
  Polygon* p_array[2];
  Circle c(3.0);  // radius = 3
  Square s(4.0);  // width = 4
  p_array[0] = &c;
  p_array[1] = &s;

  double sum_area = 0;
  for (int i=0;i<2;i++) {
     p_array[i]->print() ;
     sum_area += p_array[i]->area();
  }
  printf("Total area = %f\n", sum_area);
  return 0;
}
//------------------area.cpp : end-------------------------------------
```

The Circle class is derived from the abstract base class Polygon, and has the following definition:

```
//------------------circle.h : begin ----------------------------
#ifndef CIRCLE_H
#define CIRCLE_H

#include "polygon.h"

class Circle : public Polygon {
  double radius;
public:
  Circle (double in_radius) { radius = in_radius; }
  virtual double area () { return 3.14159*radius*radius; }
  virtual void print () { printf ("Circle(%f)\n",radius);}
};
#endif
//------------------circle.h : end------------------------------
```

(a) Define `polygon.h` and implement the member functions inside the class package if there are any. You can score full credit only if your abstract class is a minimal one that allows the main program shown here to work.

Answer:

```
//-------------------------polygon.h : begin---------------------
#ifndef POLYGON_H
#define POLYGON_H

class Polygon {
  public:
    virtual double area () = 0;
    virtual void print() = 0;
};
#endif
//-------------------------polygon.h : end ----------------------
```

(b) Define `square.h` and implement the member functions inside the class package if there are any. You can score full credit only if your derived class is a minimal one that allows the main program shown here to work.

Answer:

```
//--------------------square.h : begin-------------------------
#ifndef SQUARE_H
#define SQUARE_H

#include "polygon.h"

#include <stdio.h>

class Square : public Polygon {
  double width;
public:
  Square (double in_width): width(in_width) { }
  double area () { return width*width; }
  void print() { printf ("square(%f)\n", width); }
};
#endif
//--------------------square.h : end---------------------------
```

(c) What will happen if all the instances of "virtual" (in `polygon.h`, `square.h`, and `circle.h`) were removed from the program? Explain briefly.

Answer:

Program cannot be compiled.

Non-virtual functions cannot be initialized to zero.

(d) Rather than removing all the instances of "virtual" in the program, what will happen if we commented out the `area()` and `print()` in `circle.h`?

Answer:
Program cannot be compiled.

Circle becomes an ABC if the pure virtual function is not overriden. / Objects cannot be instantiated from an ABC.

5 Static Class

Define a class called `Singleton` that cannot be instatiated more than once. That is, a program that uses the Singleton class during execution can have *either zero or one instance* of Singleton.
If you try to create a second instance, you would get the first instance already created. The content of the class is irrelevant for the problem. The key is that there could be no way to create more than on instance of the class.
Hints and requirements:

1. You need to declare something static to keep track of the unique instance of Singleton.
2. The default construction must be made private, so that direct construction is not allowed.
3. You do not need to define other constructors except the private default constructor.
4. Because of the private default constructor, a friend function should be defined to create the unique instance of the class, or return the instance already created.
5. You are also required to define the destructor `Singleton()`.

You can get a glimpse of the above requirement by looking at the sample program that uses the Singleton class as follows:

```
int main ()
{
  Singleton *s1, *s2;   // use pointers here because private default
                        // constructors cannot be called at initialization

  s1 = NewSingleton(); // the unique instance of Singleton is created
  s2 = NewSingleton(); // attempt to create another instance of Singleton
  if (s1==s2)
    puts ("same instance!");
  else
    puts ("bug!!");
  return 0;
}
```

Write your answer on the next page.

Write your answer here.

Answer:

```
class Singleton {
private:
  static Singleton* instance;
  Singleton() {} // must have constructor private so no (default) direct
                 // construction is allowed
 // public constructor function to construct instance
public:
  friend Singleton* NewSingleton();
  ~Singleton();
};

Singleton* Singleton::instance = NULL;

Singleton::~Singleton() {
  if (Singleton::instance != NULL) {
    delete Singleton::instance;
    Singleton::instance = NULL;
  }
}


Singleton* NewSingleton() {
  if (Singleton::instance == NULL)
    Singleton::instance = new Singleton();
  return Singleton::instance; // return the unique instance
}
```

6 Function Pointers and Function Objects

(a) Consider the following program and its output:

```
//--------------main.cpp--------------------
#include <iostream.h>
#include "template.h"

int hash_func_int (const int& a, const int& b) {
  cout << "hash function for int is used" << endl;
}

double hash_func_double (const double& a, const double& b) {
  cout << "hash function for double is used" << endl;
}

int main() {
  Hashtable<int>     int_hash;
  Hashtable<double>  double_hash;

  Hashtable<int>::Funcptr int_fptr;
  Hashtable<double>::Funcptr double_fptr;

  int_hash.set_function_pointer (hash_func_int);
  int_fptr = int_hash();
  int_fptr(10,20);
  double_hash.set_function_pointer (hash_func_double);
  double_fptr = double_hash();
  double_fptr(12.3, 40.5);

  return 0;
}
```

The output of the program is:

```
hash function for int is used
hash function for double is used
```

Your task is to code up `template.h` so that the program can compile and run, that is, to define the class template `Hashtable`. Note from the main program that objects instantiated from this class are **function objects**, where an assignment of **function pointer** is involved.

Requirement: You should also notice that a `typedef` statement should be included inside the `Hashtable` class, which is where `Funcptr` is defined.

Write your answer on the next page.

Answer:

```
//--------------template.h--------------------
template <class T>

class Hashtable {
typedef T (*Funcptr) (const T&, const T&);
// note the correct placement of parentheses for (*Funcptr)

private:
  Funcptr funcPointer;
// alternative solution:
// T (*funcPointer) (const T&, const T&);

public:
  void set_function_pointer (Funcptr f) { funcPointer = f; }
  Funcptr operator()() { return funcPointer; }
};
```

(b) Consider the following definition:

```
int (*pfunc)(float, char, int);
```

(1) What does this definition mean?
Answer:
A variable that is a pointer to the function with three arguments, `float`, `char`, `int` and return an `int`.
(2) What does it mean if the first pair of parentheses are omitted, that is

```
int *pfunc(float, char, int);
```

Is it the same as (1)? If it is not, explain the new meaning.
Answer:
It is not the same as (1). It now declares a function with three arguments, `float`, `char`, `int` and returns a pointer of `int`.

7 Container Class

This question concerns with the design and implementation of a container class `Array`, where you should apply what you learned about templates and operator overloading.
Note that your implementation should be free of any dangling pointers/references or memory leaks in order to avoid mark deduction.
Code up `array.h` so that the following main program can compile. The sample input and output are shown.

```
//------------------------main.cpp---------------------------------
#include <iostream>
#include "array.h"
using namespace std;

int main()
{
  int i, size;
  cout << "Enter size : "; cin >> size;
  Array<char> a(size);
  // ASCII code for 'A' is 65.
  for (i=0;i<6;i++) { a[i] = i+65; cout << (char)(a[i]) << " " ; }
  cout << endl;

  Array<char> b(a);
  for (i=0;i<size;i++) { cout << (char)(b[i]) << " "; }
  cout << endl;
  cout << ((a.getsize() == b.getsize()) ? "true" : "false") << endl;
  cout << a[100] << endl;  // out of bound
  b[10000] = 'j';          // out of bound
  for (i=0;i<size;i++) {  // the array b is unchanged
   cout << (char)(b[i]) << " ";
  }
  cout << endl;

  Array<char> c(10), d(10);
  d = c = b;  cout << "cascading assignment..." << endl;
  for (i=0;i<d.getsize();i++) d[i] += 1;
  for (i=0;i<d.getsize();i++) cout << d[i] << " "; cout << endl;
  for (i=0;i<c.getsize();i++) c[i] += 2 ;
  for (i=0;i<c.getsize();i++) cout << c[i] << " "; cout << endl;
  for (i=0;i<b.getsize();i++) cout << b[i] << " ";
  cout << "<--- b remains unchanged..." << endl;

  return 0;
}
```

Sample input and output:

```
Enter size : 4
A B C D array index 4 out of bound - value UN trustworthy
array index 4 out of bound - value UN trustworthy
E array index 5 out of bound - value UN trustworthy
array index 5 out of bound - value UN trustworthy
F
A B C D
true
```

```
array index 100 out of bound - value UN trustworthy
F
array index 10000 out of bound - value UN trustworthy
A B C D
cascading assignment...
B C D E
C D E F
A B C D <--- b remains unchanged...
```

Write your answer on this and the next page.

```
#ifndef ARRAY_H
#define ARRAY_H

template <typename T>
class Array {
public:
   Array (int = 10);
   Array (const Array &);
   T& operator[](int i);
   const T& operator[](int i) const;
   Array& operator= (const Array &);
   int getsize() const { return _size; };
   ~Array();
private:
   T* _data;
   int _size;
};

template<typename T>
Array<T>::Array (int size) : _size(size) {
  _data = new T[size];
  if (!_data) {
    cerr << "Memory Error" << endl;
    exit(1);
  }
}

template<typename T>
Array<T>::Array (const Array<T> &a) : _size (a._size) {
  _data = new T[a._size];
  if (!_data) {
    cerr << "Memory Error" << endl;
    exit(1);
  }
  memcpy(_data,a._data,_size*sizeof(T));
}

template<typename T>
Array<T>::~Array () {
  if (_data) {
    delete [] _data;
    _data = NULL;
    _size = 0;
  }
}
```

This is a blank page.

```
template<typename T>
T& Array<T>::operator[](int i) {
  static T tmp = 'Z';
  if ((i < 0) || (i >= _size)) {
    cerr << "array index " << i << " out of bound - value UN trustworthy" << endl;
    return tmp; // return a dummy reference
  } else
    return _data[i];
}

template<typename T>
const T& Array<T>::operator[](int i) const {
  static T tmp = 'Z';
  if ((i < 0) || (i >= _size)) {
    cerr << "array index " << i << " out of bound - value UN trustworthy" << endl;
    return tmp; // return a dummy value
  } else
    return _data[i];
}

template<typename T>
Array<T>& Array<T>::operator= (const Array &a) {
  _size = a._size;
  if (_data) delete [] _data;    // free the old memory if it exists
  _data = new T[a._size];
  if (!_data) {
    cerr << "Memory Error" << endl;
    exit(1);
  }
  memcpy(_data,a._data,_size*sizeof(T));
  return *this;
}

#endif
```

8 STL – Convex Hull

You are required to use STL in this problem to simplify your coding. Assume the header files has been included.
In this problem, let us consider a fundamental structure in computational geometry, called the convex hull (Figure 1). Given a set of points, the convex hull is defined intuitively by surrounding a collection of points with a rubber band and letting the rubber band snap tightly around the points.

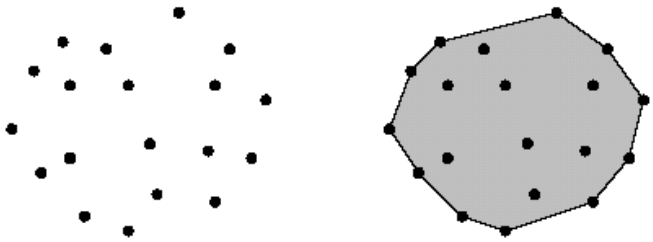


Figure 1: convex hull

The (planar) convex hull problem is, given a set of n points P in the plane, output a representation of P's convex hull. The convex hull is a closed convex polygon, the simplest representation is a counterclockwise enumeration of the vertices of the convex hull. (A clockwise is also possible. We usually prefer counterclockwise enumerations, since they correspond to positive orientations, but obviously one representation is easily converted into the other.) Ideally, the hull should consist only of extreme points, in the sense that if three points lie on an edge of the boundary of the convex hull, then the middle point should not be output as part of the hull.
In this problem, the following classes and function headers are given, and you are allowed to use them in advance.

```
#define PI 3.1415926535897931

//-- Class representing a point in 2D, (x,y)--
class Point{ public:
    Point():x(0),y(0){};
    Point(double ix, double iy):x(ix),y(iy){};
    Point(const Point &p):x(p.x),y(p.y){};

    bool operator==(const Point & p){
        return (x == p.x) && (y == p.y);
    }

    double x;
    double y;
};



//-- Class representing a line in 2D, a*x + b*y + c = 0--
class Line{ public:
    Line():a(0),b(0),c(0){};
    Line(double ia, double ib, double ic):a(ia),b(ib),c(ic){};
    Line(const Line &p):a(p.a),b(p.b),c(p.c){};

    bool operator==(const Line & p){
```

```
        return (a == p.a) && (b == p.b) && (c == p.c);
    }

    double a;
    double b;
    double c;
};


//-- A class for finding convex hull of a set of points
//-- Input is a const reference vector<Point>, input--
//-- Output is a reference vector<Point>, output--
class Convexhull{ public:
    //-- Find the convex hull of input point --
    //This Function will be implemented by you
    void FindConvexHull(vector<Point>& input, vector<Point>& output);

private:
    // -- A function returns a directed line pq from point p to q --
    // -- The length of L is normalized, i.e., L.a*L.a + L.b*L.b + L.c*L.c == 1 --
    Line FindLine(const Point& p, const Point& q);

    // -- A function returns true only if r is on the right side of the line from p to q --
    // -- If pq is horizontal, returns true only if r is above it --
    bool isOnRight(const Point& p, const Point& q, const Point& r);

};
```

To find a convex hull, there is a simple $O(n^3)$ brute-force convex hull algorithm, which operates by considering **each ordered pair of** points (p, q), and determining whether **all** the remaining points of the set lie on the one side of the directed line pq from p to q.

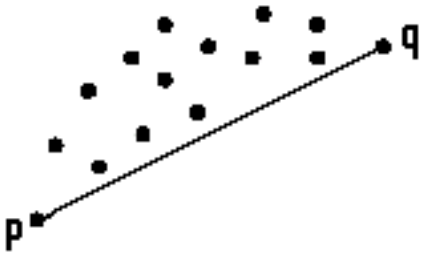


Figure 2: All other points are on one side if p and q is on convex hull

Implement the function `FindConvexHull` using the brute-force algorithm stated above. To simplify the problem, you do not need to sort the output in counterclockwise direction, and we assume that no 3 points are lying on the same straight line. (Note that the output vector should not contain redundant point)

Implement the following pseudocode of the brute force algorithm, given as follows:

```
    For Each Point p in Input
        For Each Point q Not equal to p in Input
            Find the line pq
            For Each Point r Not equal to p and q in Input
                Check if r is lying on right side of pq
            End
            If All r are lying on Right side of pq
                Push p and q into output if they are not in output
            End
        End
    End
```

Write your solution on this and the next page:

```
void FindConvexHull(vector<Point>& input, vector<Point>& output){
    vector<Point>::iterator Iitr1;
    vector<Point>::iterator Iitr2;
    vector<Point>::iterator Oitr;
    bool isOnConvexHull;
    bool isRedundant1;
    bool isRedundant2;

    for(Iitr1 = input.begin(); Iitr1 != input.end(); Iitr1++){
        for(Iitr2 = input.begin(); Iitr2 != input.end(); Iitr2++){

            if(Iitr1 != Iitr2){
                isOnConvexHull = true;
                for(Oitr = input.begin(); Oitr != input.end(); Oitr++){
                    if(Oitr != Iitr1 && Oitr != Iitr2){
                        if( !isOnRight(*Iitr1, *Iitr2, *Oitr) ){
                            isOnConvexHull = false;
                            break;
                        }
                    }
                }
            }
            else
                isOnConvexHull = false;

            if(isOnConvexHull){
                isRedundant1 = false;
                isRedundant2 = false;
                for(Oitr = output.begin(); Oitr != output.end(); Oitr++){
                    if( !isRedundant1 && *Oitr == *Iitr1 ) isRedundant1 = true;
                    if( !isRedundant2 && *Oitr == *Iitr2 ) isRedundant2 = true;
                    if(isRedundant1 && isRedundant2) break;
                }

                if(!isRedundant1) output.push_back(*Iitr1);
                if(!isRedundant2) output.push_back(*Iitr2);
```

```
                }
            }
        }
    }
```